UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

TRABAJO DE FIN DE GRADO

DEPARTAMENTO DE TEORÍA DE LA SEÑAL Y COMUNICACIONES

ESTUDIO COMPARATIVO DE DIFERENTES SOLVERS EN UN SOFTWARE DE ELEMENTOS FINITOS

Autor: Carlos Romero Castro

Tutor: Luis Emilio García Castillo

Director: Adrián Amor

Leganés, Septiembre de 2015

Título: ESTUDIO COMPARATIVO DE DIFERENTES SOLVERS EN UN SOFTWARE DE ELEMENTOS FINITOS
Autor: Carlos Romero Castro
Director: Luis Emilio García Castillo / Adrián Amor

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 8 de Octubre de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

AGRADECIMIENTOS

Después de mucho trabajo y tiempo invertido, termina una etapa importante de mi vida, es una tarea difícil agradecer a toda la gente que con su presencia a lo largo de este período me ha ayudado a llegar hasta aquí.

En primer lugar, quiero darles las gracias a mis tutores, Adrián y Luis, por el esfuerzo, la dedicación y la paciencia que han demostrado conmigo en todo momento durante la realización de este trabajo.

A mis padres y hermano, por todo su cariño, ánimo, comprensión y tantas cosas que me han dado durante toda mi vida que no se pueden describir con palabras.

A toda la gente con la que he coincidido durante mi estancia en la universidad, los que ya terminaron, y los que aún están por terminar, todos ellos buenos amigos con los que he compartido grandes momentos, de tristeza y frustración, pero sobre todo de alegría. Momentos que ya forman parte de mi vida.

A mis amigos de toda la vida y a los nuevos, gracias por estar ahí en los buenos y en los malos momentos dándome el apoyo necesario para llegar a este momento.

A todos ellos, gracias.

RESUMEN

El presente Trabajo Fin de Grado es un estudio comparativo entre distintos *solvers* aplicables en un *software* de Elementos Finitos, HOFEM (High Order Finite Element Method) desarrollado en la Universidad Carlos III de Madrid.

A través del encapsulamiento de código FORTRAN disponible, se espera, mediante una interfaz del lenguaje de programación Python obtener la solución de un sistema de ecuaciones proporcionado por las matrices obtenidas a partir de HOFEM, para, posteriormente, entregar esa solución al *software* de HOFEM para la obtención de la solución del problema en la fase de post-proceso.

Debido a diferentes problemas obtenidos durante el proceso del encapsulado del código FORTRAN, y dado el tiempo limitado de ejecución del proyecto, se optó por otra vía de acción, implementando una interfaz basada en el paso de mensajes entre lenguajes, consistente en escribir las matrices a disco, para finalmente realizar un Estudio Comparativo entre los diferentes solvers utilizados.

Palabras clave: FORTRAN, Python, FEM, HOFEM, solver

ABSTRACT

This Final Project is a comparative study between different solvers applicable to a Finite Element software, HOFEM (High Order Finite Element Method) developed at the Carlos III University of Madrid.

Through the encapsulation of available FORTRAN code, it is expected, through an interface of the programming language Python to obtain the solution of a system of equations provided by the matrix obtained from HOFEM to, then deliver that solution to HOFEM software to obtain the solution of the problem in the step of post-processing.

Due to various problems obtained during the encapsulation of FORTRAN code, and given the limited Project execution time, we chose another course of action, implementing an interface based on message passing between languages, consisting in writing matrices to disk, to finally make a comparative study between the different solvers used.

Keywords: FORTRAN, Python, FEM, HOFEM, solver

Índice general

ÍNDICE GENERAL ..... **IX**
ÍNDICE DE FIGURAS ..... **XI**
ÍNDICE DE TABLAS ..... **XIV**
INTRODUCCIÓN **1**
1.1 Introducción 1
1.2 Estructura del documento 4
ENTORNO DE TRABAJO **6**
2.1 Lenguaje de programación FORTRAN 6
2.2 Lenguaje de programación PYTHON 10
2.3 Desarrollos en Computación Numérica 13
2.3.1 NumPy *13*
2.3.2 SciPy *13*
2.3.3 Petsc4Py *14*
2.4 F2PY 16
2.4.1 Principales Características *17*
2.4.2 Evolución de F2PY *18*
2.5 FEM 20
2.5.1 Código HOFEM *21*
2.6 SOLVERS 23
2.6.1 Solvers Directos *23*
2.6.2 Solvers Iterativos *25*
METODOLOGÍA DE TRABAJO **29**
3.1 Paso0 – Funcionamiento de F2PY 30
3.2 Paso1 – Llamadas a módulos 32
3.3 Paso2 – Uso de tipos REAL (KIND) 35
3.4 Paso3 – Uso de allocatable arrays 37
3.5 Paso4 – Interconexión con librerías de HOFEM 39
ESTUDIO COMPARATIVO ENTRE SOLVERS. **43**
4.1 Paso0 – Funcionamiento de Petsc4Py 45
4.1 Paso1 – Tratamiento de las matrices HOFEM desde Python 48
4.2 Paso2 – Resolución del problema de Elementos Finitos mediante Petsc4Py 50
4.3 Comparación entre distintos solvers para un problema de Elementos Finitos. 52
CONCLUSIONES Y LÍNEAS FUTURAS **56**
5.1 Conclusiones 56
5.2 Futuras Líneas de Implementación 58
5.3Herramientas Objeto de Estudio 59
5.2.1 f2py_wrapper_gen – Encapsulado de Código Fuente de FORTRAN 90 ..... *59*
5.2.2Sfepy: Simple Finite Elements in Python *59*
PRESUPUESTO **62**
PLANIFICACIÓN DEL PROYECTO **63**
GLOSARIO **64**
REFERENCIAS **65**
BIBLIOGRAFÍA **67**

Índice de figuras

FIGURA 1. DIAGRAMA DE BLOQUES INTERCONEXIÓN PYTHON-HOFEM 4

FIGURA 2. CÓDIGO FORTRAN EN UNA TARJETA PERFORADA, MOSTRANDO EL USO ESPECIALIZADO DE LAS COLUMNAS 1-5,6 Y 73-80 8

FIGURA 3. LENGUAJES DE PROGRAMACIÓN MÁS UTILIZADOS DURANTE 2014 11

FIGURA 4. CRONOLOGÍA DE PYTHON - LAS VERSIONES INDICADAS EN ROJO SE CONSIDERAN OBSOLETAS..12

FIGURA 5. SOLICITUDES DE EXTRACCIÓN EN EL REPOSITORIO GITHUB POR LENGUAJES 13

FIGURA 6. LOGOTIPO DE SCIPY 16

FIGURA 7. PYTHON VS FORTRAN 18

FIGURA 8. ATRIBUTOS Y DECLARACIONES FORTRAN COMPATIBLES CON F2PY.....20

FIGURA 9. VISUALIZACIÓN DE UNA SIMULACIÓN FEM DE LA DEFORMACIÓN DE UN COCHE TRAS UN CHOQUE FRONTAL ASIMÉTRICO.....22

FIGURA 10. GRÁFICA DE CONVERGENCIA PARA UN MÉTODO ITERATIVO 28

FIGURA 11. CÓDIGO 'HOLA MUNDO' EN PYTHON.....32

FIGURA 12. EJECUCIÓN DEL MÓDULO 'ASTERISCOS'.....34

FIGURA 13. CÓDIGO FORTRAN DEL MÓDULO 'VECTORES'.....35

FIGURA 14. EJECUCIÓN DEL MÓDULO 'VECTORES'.....35

FIGURA 15. USO DE F2CMAP 37

FIGURA 16. USO DE F2CMAP (II).....37

FIGURA 17. CÓDIGO FORTRAN DEL MÓDULO 'ALLOC'.....38

FIGURA 18. ERROR ALLOCATABLEARRAYS.....39

FIGURA 19. DOCUMENTACIÓN DE LA FUNCIÓN SOBRE ALLOCATABLEARRAYS EN PYTHON 39

FIGURA 20. EJECUCIÓN DEL MÓDULO SOBRE ARRAYSALLOCATABLES EN PYTHON 39

FIGURA 21. SOLUCIÓN MEDIANTE EL MÉTODO DEL CONJUGADO GRADIENTE 44

FIGURA 22. SOLUCIÓN MEDIANTE EL MÉTODO GMRES 45

FIGURA 23. TIEMPOS DE EJECUCIÓN PARA DISTINTOS SOLVERS 45

FIGURA 24. CÓDIGO PYTHON PARA LEER LAS MATRICES DE HOFEM 47

FIGURA 25. CÓDIGO PYTHON/PETSC4PY PARA LAS MATRICES DE HOFEM 48

FIGURA 26. CÓDIGO PYTHON/PETSC4PY PARA LAS MATRICES DE HOFEM (II) 49

FIGURA 27. DIAGRAMA DE BLOQUES DE LOS DISTINTOS SOLVERS DE PYTHON UTILIZADOS 50

FIGURA 28. TIEMPO DE RESOLUCIÓN PARA DISTINTOS SOLVERS Y MATRICES 51

FIGURA 29. TIEMPO DE RESOLUCIÓN PARA MATRICES MUY DISPARES EN TAMAÑO 51

FIGURA 30. SIMULACIÓN HOFEM DE UN PROBLEMA DE ELECTROMAGNETISMO 58

FIGURA 31. LOGOTIPO DE SFEPY 60

Índice de tablas

TABLA 1. COMPARACIÓN ENTRE DISTINTOS LENGUAJES DE PROGRAMACIÓN.....13

TABLA 2. ASIGNACIÓN DE TIPOS FORTRAN A TIPOS C 36

TABLA 3. MATRICES HOFEM 43

TABLA 4. TIEMPO DE RESOLUCIÓN PARA DISTINTOS SOLVERS 45

TABLA 5. TIEMPO DE RESOLUCIÓN PARA DISTINTOS SOLVERS Y MATRICES 51

Capítulo 1

Introducción

1.1Introducción

En la línea de trabajo a la que pertenecen el tutor y el director del este Trabajo Fin de Grado se ha desarrollado una *suite software* de simulación electromagnética que se ha ido mejorando gradualmente en los últimos años. Estas mejoras han sido enfocadas en los últimos años a mejorar la rapidez del *software* para competir con otros *softwares* comerciales más expandidos y generalizados como CST, HFSS o FEKO [1, 2,3].El principal cuello de botella de estos programas se encuentra en la velocidad del resolvedor o *solver* del sistema de ecuaciones que se obtiene al resolver un problema electromagnético.

El objetivo de este Trabajo Fin de Grado es la comparación entre distintos resolvedores o *solvers* de sistemas de ecuaciones en un *software* basado en el Método de los Elementos Finitos.

Este método está basado en la definición de pequeños elementos en los que se calcula localmente la solución ensamblándolos posteriormente para determinar la solución global. Esta solución se representa como una combinación lineal de un conjunto de funciones llamadas comúnmente funciones de base o funciones de

interpolación. Tiene su origen en el análisis de estructuras en ingeniería mecánica y se ha impuesto como la tecnología de referencia en una gran variedad de campos de la física e ingeniería y, en particular, es común su uso también en electromagnetismo.

De esta forma, el Método de los Elementos Finitos tiene cuatro etapas fundamentales [4]:

- Discretización del dominio: consiste en elegir cómo se va a dividir el espacio de simulación.
- Selección de las funciones de interpolación: se determina qué funciones se van a utilizar (por ejemplo, orden uno, orden dos…) para aproximar la solución del campo electromagnético.
- Formulación del sistema de ecuaciones: con la discretización del sistema del dominio y la definición en cada elemento de unas incógnitas y funciones de interpolación, se formula y se ensambla un sistema de ecuaciones donde las incógnitas son los coeficientes que multiplicarán a las funciones en esa combinación lineal.
- Solución del sistema de ecuaciones.

Para la última fase existen distintas alternativas que se desean comparar en este Trabajo Fin de Grado. HOFEM (*Higher Order Finite Element Method,* [5,6]) se encuentra desarrollado desde 2010 en lenguaje FORTRAN [7], en el que es característica una alta eficiencia computacional pero es muy poco flexible, de forma que cambiar piezas de este código es costoso. Python destaca por su flexibilidad y por tener una comunidad creciente de programadores, de forma que sus implementaciones son cada vez más eficientes y rápidas.

En este caso concreto, se quieren probar diferentes *solvers* de sistemas de ecuaciones generados por un método de los elementos finitos, y es habitual que cuenten con versiones escritas en Python y en FORTRAN. Por todo ello, es conveniente implementar una interfaz de FORTRAN a Python y así poder hacer pruebas en este último lenguaje, observando si es conveniente cambiar de un *solver* a otro en FORTRAN o utilizar los códigos de Python si son más rápidos.

CAPÍTULO 1: INTRODUCCIÓN

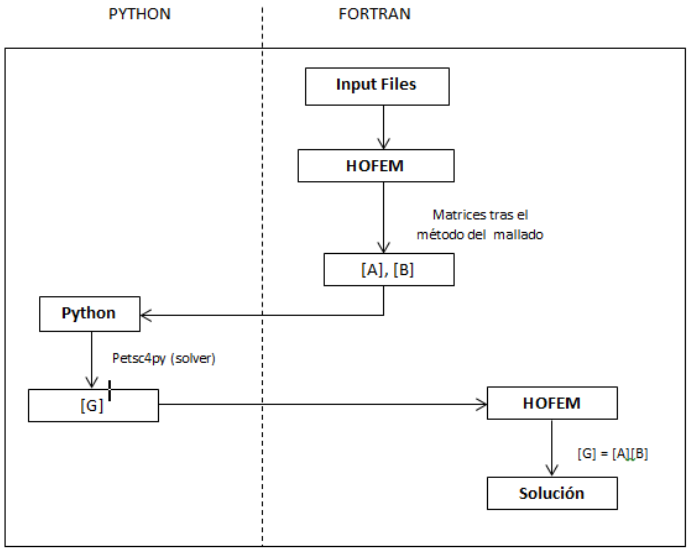


Figura 1. Diagrama de bloques interconexión Python - FORTRAN

1.2 Estructura del documento

El presente documento está dividido en 5 capítulos en lo que sigue se hace un breve resumen del contenido de los mismos.

El primero de ellos [capítulo 1] es el de introducción, dónde se expone brevemente el trabajo a realizar durante el presente proyecto así como su origen y características y sus posibles aplicaciones. En él también se encuentra la motivación del trabajo a desarrollar a lo largo de estas páginas y los objetivos que se quieren alcanzar con ello.

El [capítulo 2] está dedicado a profundizar en el entorno de trabajo en el que se desarrolló el presente Trabajo Fin de Grado. Se presentará aquí una descripción de los dos lenguajes de programación utilizados, FORTRAN y Python. El capítulo desarrolla las principales herramientas de Python en materia de computación numérica (NumPy, SciPy, Petsc4Py y F2Py). Por último, se incluye una descripción de los *solvers* utilizados atendiendo a su tipo (directo o indirecto).

La siguiente parte del presente Trabajo Fin de Grado se centrará en el desarrollo de la metodología de trabajo, recorriendo los diversos pasos recorridos durante el presente proyecto, desde la comprensión de la herramienta F2Py, hasta los errores de compilación que obligaron a estudiar vías de acción alternativas. Este [capítulo 3] contiene toda la información relativa a la primera vía de acción del presente proyecto.

En el [capítulo 4] se realizará el estudio comparativo entre los distintos *solvers* descritos en anteriores secciones de este documento, atendiendo al tiempo de ejecución de cada *solver*, el tamaño de las matrices del problema y la diferencia entre la solución analítica obtenida mediante HOFEM y la solución obtenida con Python.

Por último, en el [capítulo 5] se recogen las conclusiones y los puntos más importantes tratados en el proyecto, destacando también las posibles líneas de investigación futuras que podrían abordarse a partir de este trabajo.

Capítulo 2

Entorno de trabajo

2.1 Lenguaje de programación FORTRAN

Tal y como se ha explicado en la introducción, el código de trabajo (HOFEM) se encuentra desarrollado en FORTRAN. A continuación se describen de forma abreviada algunas particularidades de este lenguaje.

FORTRAN (FORmulaTRANslatingSystem) es un lenguaje de programación de alto nivel de propósito general, fuertemente adaptado a la computación científica y al cálculo numérico. Fue desarrollado originalmente por IBM, y usado para aplicaciones científicas y de ingeniería. FORTRAN ha estado en uso continuo durante más de medio siglo en áreas tales como la predicción numérica del tiempo, análisis de elementos finitos, dinámica de fluidos computacional, física computacional y química computacional. Es uno de los lenguajes más utilizados en el área de la computación de alto rendimiento y es el lenguaje utilizado para programas que evalúan el desempeño y el ranking de los supercomputadores más rápidos del mundo [8].

El nacimiento de este lenguaje se debe a John Backus, junto a Richard Goldberg, Sheldon F. Best, HarlanHerrick, Peter Sheridan, Roy Nutt, Robert Nelson, Irving Ziller,

Lois Haibt y David Sayre, todos ellos en la nómina de IBM, quienes en 1954 presentan el informe titulado '*Preliminary Report, Specifications for the IBM Mathematical FORmula TRANSlating System, FORTRAN*' [9]. Dos años después, en 1956, Backus y su equipo presentan a la directiva de IBM una alternativa al Lenguaje ensamblador para la computadora IBM 704.

FORTRAN se caracteriza por su potencia en los cálculos matemáticos, pero está limitado en las aplicaciones de gestión, manejo de archivos, tratamiento de cadenas de caracteres y edición de informes. Fue diseñado teniendo en cuenta el uso de la tarjeta perforada de 80 columnas, por lo que el orden de las instrucciones debía ser secuencial. Por lo tanto, la programación de los algoritmos era lineal, de forma que, para producir cualquiera alteración del orden de la lógica, se introduce la instrucción *Goto.* Debido al desarrollo de los métodos de programación, FORTRAN también fue evolucionando de versión en versión incorporando nuevas funciones, estructuras de control y compiladores. Por estas razones FORTRAN no es muy usado fuera de los campos científicos y del análisis numérico, pero permanece como el lenguaje más utilizado para desempeñar tareas de computación numérica de alto rendimiento.

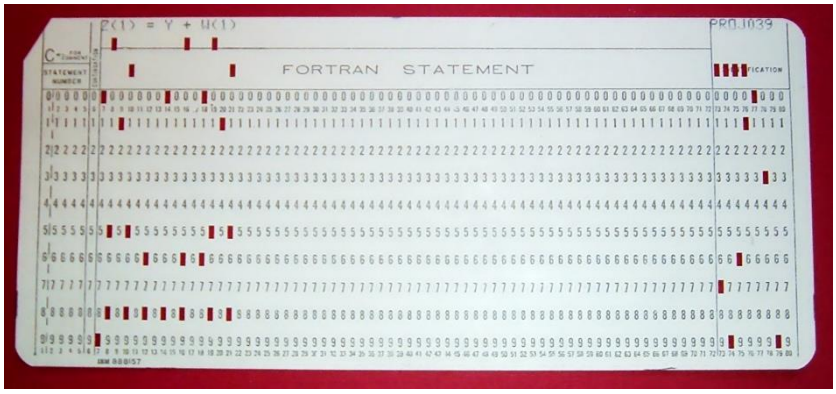


Figura 2. Código Fortran en una tarjeta perforada, mostrando el uso de las columnas 1-5, 6 y 73-80.

A lo largo de su historia han aparecido diferentes versiones, entre las que destaca la adoptada en 1966 por el ANSI (*American NationalStandardsInstitute*), en la que se definieron nuevas reglas del lenguaje y se logró la independencia respecto a la máquina, es decir, comenzó su portabilidad. Durante los años 60, los ordenadores se distribuían con FORTRAN 66 incorporado, lo que permitía escribir programas en FORTRAN en cualquier sistema y trasladarlos a otros con la seguridad de que pudieran trabajar igual que en el sistema original. Esta versión se denominó FORTRAN IV o FORTRAN 66 y su uso estaba tan extendido que se convirtió en el primer lenguaje de programación en ser regularizado oficialmente en 1972.

En 1977, apareció una nueva versión llamada FORTRAN V o FORTRAN 77. Está reflejada en el documento *ANS X3.9-1978: Programming Language Fortran*, y define dos niveles del lenguaje denominados FORTRAN 77 completo y FORTRAN 77 básico, siendo el segundo un subconjunto del primero. Se incluían instrucciones para el manejo de

cadenas de caracteres y archivos, así como otras para la utilización de técnicas de programación estructurada. FORTRAN 77 facilita la programación estructurada con bloques *"IF (…) THEN/ELSE/ENDIF".*

Actualmente existen varias versiones normalizadas del lenguaje:

- ANSI X3.198-1992 (R1997). Título: *Programming Language "Fortran" Extended*. Conocida como FORTRAN 90, se trata de un estándar publicado por ANSI. Esta nueva versión incorpora los módulos, la recursividad y sobrecarga de operadores junto con nuevos tipos de datos. Se considera una actualización importante para poner a FORTRAN al nivel de otros lenguajes modernos.

- ISO/IEC 1539-1:1997. Título: *Information technology – Programming languages – FORTRAN – Part 1: Based language.* Conocido como FORTRAN 95. En esta versión se declaran obsoletas algunas instrucciones y se incorporan conceptos propios de otros lenguajes y estilos de programación, como los archivos tipo *free-form (*más de 80 caracteres por línea*)*. Actualización también adoptada por ANSI.

- ISO/IEC JTCI/SC22/W65 N1601. Título: *Information technology – Programming languages – FORTRAN – Part 1: Based Language.* Conocida como FORTRAN 2003, es una revisión significativa del estándar FORTRAN 95, que incluye nuevas características cómo la parametrización de tipos derivados, la interoperabilidad con C y la integración por sistema operativo (argumentos de línea de comandos y variable de entorno). Introduce la Programación Orientada a Objetos (extensión del tipo y la herencia, el poliformismo, la asignación de tipos dinámicos y los procedimientos 'type-bound') [10].

- ISO/IEC JTC 1/SC 22/WG 5/N1830. Título: *Information technology – Programming languages – FORTRAN – Part 1: Based Language.* Conocida como FORTRAN 2008 [11], es una revisión menor del estándar FORTRAN 2003 que incluye nuevas características cómo la creación de sub-módulos, construcción de bloques y la sentencia *exit*.

A continuación se expone una comparativa entre FORTRAN y otros lenguajes de programación utilizados para computación científica (Python, C++, C, y Matlab), atendiendo a diversas características como la detección de errores, la optimización o la eficiencia.

Detección de errores

- Detección de errores estática únicamente en FORTRAN y C++. En Python y Matlab se comprueban dinámicamente.
- La detección de errores dinámica es el problema principal: Python y FORTRAN tienen buenas prestaciones, después Matlab. La mayoría de versiones de FORTRAN y C++ tienen prestaciones pobres o malas.

Optimización/Eficiencia

- Las prestaciones son similares usando librerías o módulos de alto nivel. A bajo nivel, C++ y FORTRAN son mucho más rápidos.
- FORTRAN es mucho más optimizable nivel máquina que C++, ya que este último debe indexar a través de múltiples archivos.
- Para la mayoría de programas basados en arrays, FORTRAN es el lenguaje más rápido. Para programas basados en punteros o caracteres, habitualmente C++. Las diferencias suelen ser generalmente marginales entre ambos.

Ingeniería de Software

- FORTRAN tiene las mejores especificaciones para trabajos en el ámbito de la computación numérica, siendo éstas, en su mayor parte, explícitas, precisas y necesarias para la portabilidad y la fiabilidad.
- Tanto FORTRAN como Python tienen módulos capaces de recopilar datos relacionados, funciones e interfaces conjuntamente.
- Python y C++ tienen excepciones, principalmente útiles para la recuperación de recursos y similares, así como Matlab que incorpora un mecanismo *try-catch* para este cometido. FORTRAN no las incorpora.

2.2 Lenguaje de programación PYTHON

Python es un lenguaje de programación multipropósito de alto nivel cuya filosofía de diseño enfatiza la productividad del programador y la legibilidad del código. Tiene un núcleo sintáctico minimalista con unos pocos comandos básicos y una semántica simple. Para muchas de las funciones a nivel del sistema operativo, contiene además por defecto una extensa y variada librería, que incluye una Interfaz de Programación de Aplicaciones. El código Python, aunque minimalista, define objetos incorporados como listas enlazadas (*list*), tuplas (*tuple*), tablas hash (*dict*), y enteros de longitud arbitraria (*long*).

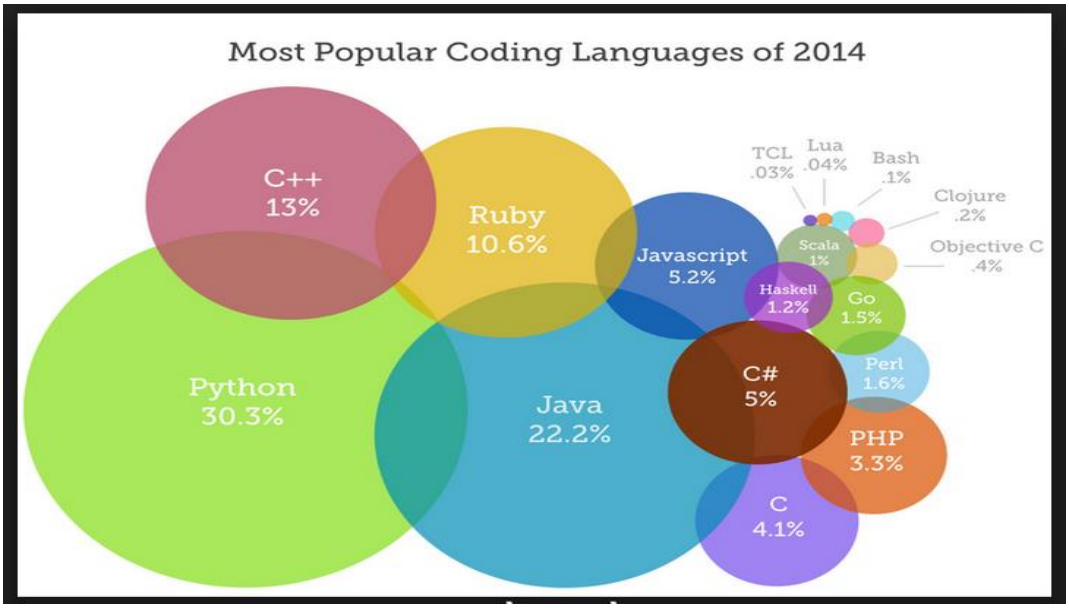


Figura 3. Lenguajes de programación más utilizados en 2014

El origen de Python se remonta a finales de la década de los 80 [12], cuando su creador, Guido Van Rossum lo diseñó en el Centro para las Matemáticas y la Informática (*CWI, CentrumWiskunde&Informatica*), con sede en Amsterdam, Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.

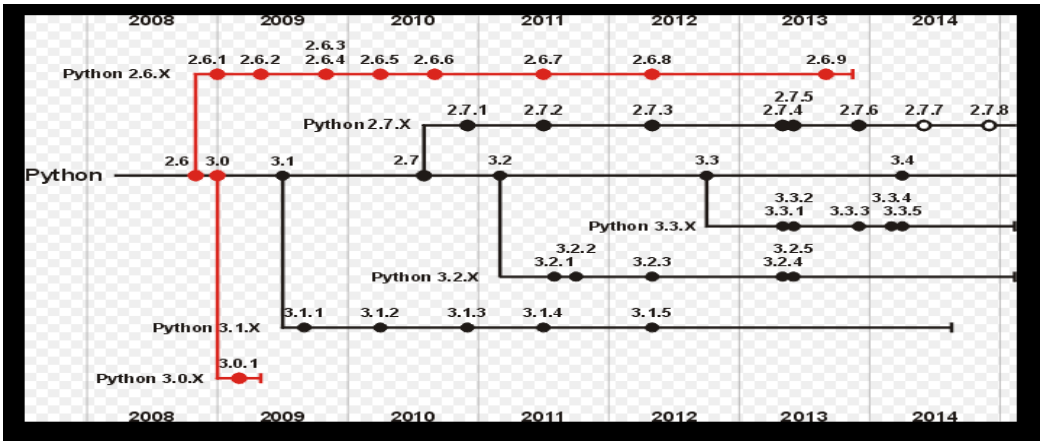


Figura 4. Cronología de Python – las versiones coloreadas en rojo se consideran obsoletas.

Python plantea un modelo abierto de desarrollo administrado por la *Python Software Foundation*, creada en marzo de 2001. Entre las actividades de esta organización destacan el desarrollo y distribución oficial de Python, la gestión de la propiedad intelectual del código y documentos realizados, así como la organización de conferencias y eventos dedicados a poner en contacto a todas aquellas personas interesadas en este lenguaje de programación. Posee una licencia de código abierto, denominada *Python Software Foundation License* [13], que es compatible con la licencia pública general de *GNU* a partir de la versión 2.1.1, e incompatible con ciertas versiones anteriores.

Se trata de un lenguaje de programación multi-paradigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

- **Tipado Dinámico:** Esta característica hace referencia a que no es necesario declarar el tipo de dato que va a contener una determinada variable. Se determinará en tiempo de ejecución según el tipo del valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo.

- **Multiplataforma:** El intérprete de Python está disponible en multitud de plataformas (UNIX, Solaris, Windows, Linux, Mac OS, etc.) por lo que si no se utilizan librerías específicas de cada plataforma, los programas podrán ser utilizados en todos los sistemas mencionados sin necesidad de aplicar grandes cambios. Se trata de una característica muy importante para el presente proyecto, dada la existencia de dos versiones del código (Windows y Linux).

Una característica importante de Python es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado enlace dinámico de métodos).

Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir módulos fácilmente en C o C++ y es posible su inclusión en aplicaciones que necesitan una interfaz programable.

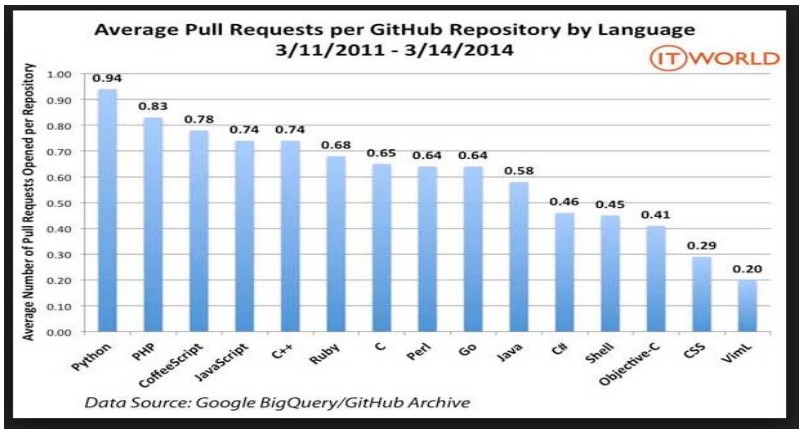


Figura 5. Solicitudes de Extracción en el Repositorio GitHub por lenguajes

Python es comparado a menudo con otros lenguajes interpretados como Java, JavaScript, Perl, TCL o Smalltalk. Aunque la mayoría de las comparaciones se concentran solamente en cuestiones lingüísticas, en la práctica la elección de un lenguaje de programación es a menudo dictada por otras limitaciones del mundo real tales como el coste, la disponibilidad, la formación e inversión previa, etc. En el caso del presente proyecto, se eligió Python debido a la existencia de numerosos desarrollos en el campo de la computación numérica, los cuales serán vistos en detalle más adelante.

Lenguaje	Uso Intencionado	Imperativo	Orientado a Objetos	Otros Paradigmas	Estandarizado
Python	Aplicación Web, scripting, computación científica.	Sí	Sí	Orientado a aspectos	No
FORTRAN	Aplicación, computación numérica.	Sí	Sí (a partir de FORTRAN 2003)		1966, ANSI 66, ANSI 77, MIL-STD-1753, ISO 90, ISO 95, ISO 2003, ISO/IEC 1539-1:2010
Java	Aplicación, negocio, client-side, desarrollo móvil, Web.	Sí	Sí	Concurrente	Estándar de facto vía *Java Language Specification*
C	Aplicación, sistemas, operaciones de nivel bajo	Sí			1989, ANSI C89, ISO C90, ISO C99, ISO C11
C++	Aplicación, sistemas.	Sí	Sí		1998, ISO/IEC 1998, ISO/IEC 2003, ISO/IEC

Tabla 1. Comparación entre distintos lenguajes de programación

2.3 Desarrollos en Computación Numérica

La elección de Python durante el presente Trabajo Fin de Grado es debida a la existencia de numerosos desarrollos en el campo de la computación numérica, los cuales se detallan a continuación.

2.3.1 NumPy

NumPy [14] es el paquete fundamental para la computación científica en Python. Se trata de una librería de Python que proporciona objetos para arrays multidimensionales, diversos objetos derivados (tales como matrices y arrays enmascarados), y una variedad de rutinas para operaciones rápidas sobre matrices, incluyendo matemática, lógica, manipulación de tamaño, clasificación y selección, así como transformadas de Fourier discretas, álgebra lineal básica y operaciones estadísticas básicas.

En el núcleo del paquete NumPy, se encuentra el objeto *ndarray.* Este encapsula arrays n-dimensionales de tipos de datos homogéneos, con muchas operaciones que se realizan sobre código compilado por cuestiones de rendimiento. Existen diferencias importantes entre las matrices de NumPy y las secuencias estándar de Python:

- Los arrays de NumPy tienen un tamaño fijo al crearse, a diferencia de las listas de Python (que pueden crecer dinámicamente). Cambiar el tamaño de un *ndarray* creará una nueva matriz y borrará la original.
- Los elementos de un array de NumPy están obligados a ser del mismo tipo, lo que significa que tendrán el mismo tamaño en memoria. Con una excepción, en los arrays de objetos se permite la inclusión de elementos de distintos tipos.
- Los arrays de NumPy facilitan operaciones matemáticas de un gran número de datos. Por lo general, este tipo de operaciones se ejecutan de manera más eficiente y con menos código del que es posible utilizando secuencias generadas en Python.

2.3.2 SciPy

SciPy [14] es un conjunto de algoritmos matemáticos construidos sobre la extensión Numpy de Python. Proporciona al usuario comandos de alto nivel y clases para manipular y visualizar datos. Mediante el uso de SciPy una sesión de Python se transforma en un entorno de procesamiento de datos que rivaliza con otros sistemas como Matlab, Octave o SciLab.

Figura 6. Logotipo de SciPy

El hecho de que SciPy esté basado en Python proporciona un beneficio adicional, ya que éste es un lenguaje de programación disponible para su uso en el desarrollo de programas sofisticados y aplicaciones especializadas. Las aplicaciones científicas que utilizan SciPy obtienen un gran beneficio, gracias al desarrollo de módulos adicionales en diversos entornos por desarrolladores software de todo el mundo. Todo, desde la programación paralela a la web, hasta las bases de datos de las subrutinas y las clases se ha puesto a disposición del programador de Python. Toda esta información está disponible además de las librerías matemáticas en SciPy.

2.3.3 Petsc4Py

PETSc (*Portable Extensible ToolkitforScienticComputation*) [15] es un conjunto de estructuras de datos y rutinas para la solución escalable de aplicaciones científicas modeladas por ecuaciones diferenciales parciales. El uso de PETSc está especialmente difundido para cálculos científicos, álgebra lineal paralela, con *solvers* tanto lineales como no lineales. Las principales ventajas de PETSc son:

- ToolKit: Contiene *solvers* de alto nivel, pero también herramientas de bajo nivel para trabajar de forma independiente.
- Portátil: PETSc está disponible en cualquier plataforma que pueda incluir MPI.

Petsc4Pyes la implementación de Python para PETSc. Su principal característica en el ámbito de la computación numérica es que las matrices no son vectores (como ocurre en Python). Los elementos con los que es posible trabajar desde Petsc4Py son:

- Establecimiento de índices: permutaciones, indexación en vectores y re numeración.
- Vectores: secuenciales y distribuidos.
- Matrices: secuenciales y distribuidas. Dispersas y densas.
- *Solvers* lineales: métodos de Krylov.
- Precondicionadores: *solvers* directos.

- Solucionadores no lineales: búsqueda en línea, región de confianza, *matrix-free.* En computación matemática, un método *matrix-free* es un algoritmo para la resolución de un sistema de ecuaciones o un problema de valores propios que no almacena una matriz de coeficientes explícitamente, pero accede a la matriz mediante la evaluación de productos matriz-vector. Estos métodos pueden ser preferibles cuando la matriz es tan grande que su almacenamiento y manipulación costaría una gran cantidad de memoria y tiempo computacional, incluso con la utilización de métodos para matrices dispersas.

El uso de Petsc4Py es muy similar al de otras herramientas de Python. Proporciona todas las funcionalidades de PETSc en una interfaz al estilo de Python, permite la gestión de toda la memoria utilizada durante el proceso, tanto su creación como su destrucción, y la visualización de resultados mediante MatPlotLib (la librería de Python que permite la representación de figuras de calidad en una gran variedad de formatos impresos y entornos interactivos en diversas plataformas, con la posibilidad de uso tanto en scripts de Python, como en servidores de aplicaciones web).

2.4 F2PY

Python es cada vez más utilizado en el ámbito de la computación numérica [16] (reflejado en el desarrollo de diversas herramientas para este entorno que ya se han descrito.) debido a las características explicadas en el apartado 2.2. Por ello, puede servir tanto de lenguaje de control para el manejo de programas existentes, como de lenguaje 'puente' para la combinación de diferentes sistemas. Además, Python ofrece una gran colección de módulos, una gran comunidad de usuarios establecida, y una gran variedad de documentación en forma de libros y referencias en línea.

Existe una gran variedad de herramientas bien conocidas en este ámbito adaptadas a Python: *MPI4PY (*MessagePassing Interface forPython*)*, *PYMUMPS* (MultifrontalMassivelyParallelSparseDirectSolver) y *F2PY (*FORTRAN to Python interface generator*)*, de la que se hablará a continuación [17,18].

F2PY (FORTRAN to Python interface generator) es un proyecto cuyo objetivo es proporcionar una conexión entre los lenguajes Python y FORTRAN. F2PY es una herramienta de Python para la creación de módulos C/API de este lenguaje a partir de ficheros de cabecera (tanto escritos a mano como generados por F2PY o directamente de fuentes FORTRAN). Los módulos generados facilitan:

- Llamadas a FORTRAN 77/90/95, módulos de FORTRAN 90/95 y funciones de C desde Python.
- Utilizar subrutinas, datos en bloques COMMON y módulos de datos de FORTRAN 90/95 (incluyendo *allocatable arrays*) desde Python.
- Llamadas a funciones de Python desde FORTRAN o C (*callbacks*).
- Manipulación automática de la diferencia en el orden de almacenamiento de arrays multidimensionales de FORTRAN y arrays numéricos de Python.

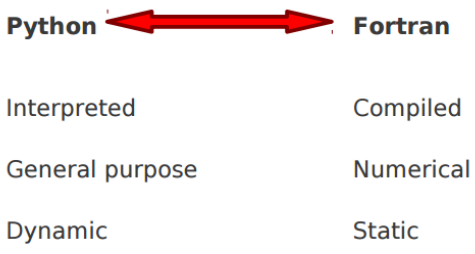


Figura 7. Python vs Fortran

F2PY es capaz de construir módulos de extensión para librerías compartidas en un solo comando. F2PY utiliza el módulo *scipy_distutils* de SciPy, el cual soporta varios de los principales compiladores de FORTRAN. Los módulos de extensión creados con F2PY son dependientes del paquete *Numpy* que proporciona menor complejidad del lenguaje para arrays multi-dimensionales en Python.

F2PY fue creado en 1999 por Pearu Peterson [19], mientras era estudiante de doctorado en la Universidad Técnica de Tallin, y en 2005 después de varias versiones estables quedó incluido dentro de *Numpy*.

2.4.1 Principales Características

F2PY escanea códigos FORTRAN reales para producir los denominados ficheros de cabecera (archivos .pyf). Estos ficheros contienen toda la información (nombres de función, argumentos y sus tipos, etc.) que se necesitan para construir enlaces Python a funciones de FORTRAN o C.

- La sintaxis de estos ficheros de cabecera está tomada de las especificaciones del lenguaje FORTRAN 90/95 y tiene algunas extensiones específicas para F2PY. Los ficheros de cabecera definen la interfaz a los subprogramas FORTRAN (o C) que se llamarán desde Python:
    - F2PY resuelve dependencias entre argumentos (esto es relevante para el orden de inicialización de variables en los módulos de extensión).
    - Los argumentos pueden ser especificados para ser opcionales u ocultos, lo que simplifica considerablemente las llamadas a FORTRAN desde Python.
    - Se puede diseñar cualquier cabecera Python para una función FORTRAN dada, por ejemplo, cambiar los argumentos de orden, presentar argumentos auxiliares, ocultar los argumentos, procesar argumentos antes de pasarlos a código FORTRAN y devolver argumentos como salidas de funciones generadas con F2PY.

- F2PY genera automáticamente *docstrings* (y opcionalmente documentación LaTeX) para los módulos de extensión.

- Las funciones generadas con F2PY aceptan objetos arbitrarios de Python como argumentos. La interfaz de F2PY se encarga automáticamente de la conversión entre tipos de datos y de la manipulación de arrays no contiguos. Es importante señalar, que en caso de que sea necesaria una conversión de este tipo, y exista una posible pérdida de información (por ejemplo, al convertir *float* a *int*), F2PY no lanza ninguna excepción. En una conversión de tipo complejo a real, solo la parte real del número complejo es utilizada.

- Las siguientes construcciones de FORTRAN son reconocidas por F2PY:
    - Todos los tipos básicos de FORTRAN.
    - Arrays multidimensionales de todos los tipos básicos.
    - Atributos y declaraciones.

```
intent([ in | inout | out | hide | in,out | inout,out | c |
         copy | cache | callback | inplace | aux ])
dimension(<dimspec>)
common, parameter
allocatable
optional, required, external
depend([<names>])
check([<C-booleanexpr>])
note(<LaTeX text>)
usercode, callstatement, callprotoargument, threadsafe, fortranname
pymethoddef
entry
```

Figura 8. Atributos y declaraciones Fortran compatibles con F2PY.

- Debido a que sólo existen pequeñas (y fácilmente manejables) diferencias entre las llamadas a funciones en FORTRAN o en C desde los módulos de extensión de F2PY, F2PY es también muy adecuado para adaptar librerías de C a Python.

F2PY tiene buenas prestaciones con FORTRAN 77 y FORTRAN 90/95, ya que no incluyen características incluidas a partir de FORTRAN 2003 como pueden ser los punteros, tipos derivados o arrays de forma indeterminada. Sin embargo, el soporte de FORTRAN 90/95 es incompleto, ya que existen errores sin solucionar (uno de los cuales impide su correcto funcionamiento en Python 3).

2.4.2 Evolución de F2PY

Desde su creación en 1999, F2PY ha experimentado un crecimiento constante, siempre enfocado hacia el objetivo inicial de su creador: la posibilidad de realizar de manera automática el encapsulado de código FORTRAN a Python.

El 9 de Julio de 1999 se escribieron las primeras líneas de esta herramienta, se hizo pública bajo el nombre '*FPIG – Fortran toPython Interface Generator*'.

En octubre de ese mismo año se reescribió gran parte de F2PY, de forma que la herramienta fuera capaz de leer código real FORTRAN y determinar la cabecera de las rutinas de FORTRAN. El principal punto de desarrollo era que F2PY fuera capaz de leer código arbitrario FORTRAN 77/90/95. Sin embargo, la generación de funciones C/API de

Python aún necesitaba de mejores prestaciones. La nueva versión de F2PY se llamó '*f2py – Fortran toPython C/API generator, the Second Edition*'.

El 25 de enero del año 2000, se anunció el lanzamiento de F2PY (versión 1.116), la herramienta de generación de funciones para el encapsulado de funciones FORTRAN estaba lista. Tenía muchas características nuevas y era más robusta que nunca. En junio de ese mismo año, se anunció la creación de un foro de discusión, para usuarios de F2PY.

Durante el período comprendido entre los años 2000-2002, se desarrollaron tres nuevas versiones de F2PY. Estas versiones incluían nuevas características como el soporte de *Win32*, la reescritura del generador de código para lograr una mayor calidad de los módulos C/API generados, la corrección de errores de tamaño en arrays multidimensionales al volver de las rutinas FORTRAN y el soporte para MAC/OS.

El 28 de Octubre de 2005, el código base de f2py fue incorporado a SciPy bajo el árbol de fuentes de NumPy (versión de f2py 2.1381). A partir de este momento, en el desarrollo de f2py se utilizan estructuras de soporte de NumPy.

En la primavera de 2006, f2py era estable y su uso estaba extendido para la encapsulación de códigos escritos en FORTRAN 77, FORTRAN 95 e incluso códigos C simples. Sin embargo, aún no existía la posibilidad de encapsular tipos derivados de Fortran 95 debido a que su implementación sería demasiado tediosa y difícil con el código base de f2py de la época.

Actualmente, el código para la encapsulación de FORTRAN 77/90/95/2000 es estable, pero se encuentra en desarrollo activo y disponible dentro de la librería NumPy.

2.5 FEM

El método de los elementos finitos (FEM) es un método numérico general para la aproximación de soluciones de ecuaciones diferenciales parciales muy utilizado en diversos problemas de ingeniería y física.

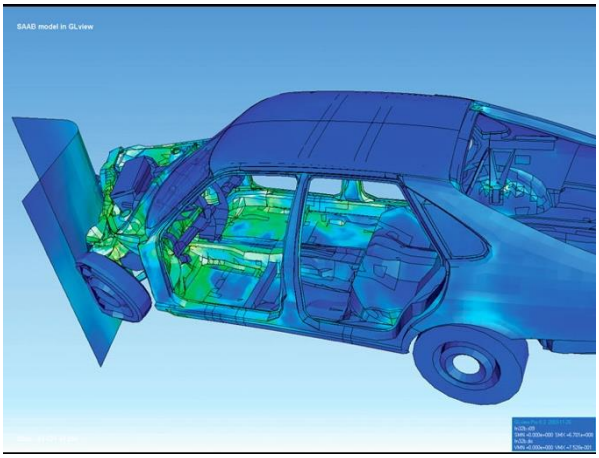


Figura 9. Visualización de una simulación FEM de la deformacion de un coche tras un choque frontal asimétrico.

En muchos problemas en el campo del electromagnetismo, así como en otras áreas de ingeniería, el resultado final de una formulación de elementos finitos es un conjunto de ecuaciones algebraicas lineales que puede ser escrito como se muestra a continuación.

$$
\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots\ldots\ldots\ldots\ldots + a_{1n}x_n = b_1 \\
&a_{21}x_1 + a_{22}x_2 + \ldots\ldots\ldots\ldots\ldots + a_{2n}x_n = b_2 \\
&\vdots \\
&a_{n1}x_1 + a_{n2}x_2 + \ldots\ldots\ldots\ldots\ldots + a_{nn}x_n = b_n
\end{aligned}
\qquad \boldsymbol{Ax} = \boldsymbol{b}
$$

Donde ***A*** es una matriz cuadrada $n \times n$, ***x*** representa el vector incógnita que debe determinarse, y ***b*** representa el vector conocido. Cuando las dimensiones de la matriz o el número de incógnitas son del orden de miles, existen diferentes algoritmos disponibles para obtener su solución de la forma más rápida posible.

Sin embargo, para problemas reales, las dimensiones de la matriz son mucho mayores. En consecuencia, cuando se busca una solución numérica con la ayuda de un

sistema digital usualmente se encuentran dos problemas: una alta demanda de memoria de computación y un largo tiempo de computación.

El primer problema se soluciona observando las propiedades especiales de la matriz. Las matrices resultantes de la discretización de elementos finitos son siempre dispersas (muchos elementos de la matriz son ceros) y simétricas. Si se pudieran explotar esas propiedades completamente, podría almacenarse una matriz con unas dimensiones superiores a varios cientos de miles de líneas.

Por ejemplo, se considera un sistema escalar de, por ejemplo, tres dimensiones con 100,000 nodos y 100,000 incógnitas. Cada nodo está conectado a 19 nodos vecinos, existiendo solamente 20 elementos de la matriz no nulos en cada línea de la matriz. Debido a la simetría, en promedio, es necesario almacenar 10 elementos por cada línea. De este modo, para un sistema de 100.000 incógnitas, sólo 10 x 10.000 elementos de una matriz de tamaño 100.000 x 100.000 tienen valores distintos de cero que necesiten ser almacenados. Esto, además de los dos vectores de enteros necesarios para indicar fila y columna de cada elemento (si se toma el sistema <coo, aij>), asciende a menos de un seis por mil del espacio de almacenamiento requerido para la matriz completa correspondiente.

Adicionalmente a la drástica reducción de demanda de almacenamiento en los computadores, las propiedades especiales de las matrices de elementos finitos pueden reducir también el tiempo de computación. No es necesario generar un gran número de elementos nulos en la matriz y, por lo tanto, las operaciones con esos elementos en el proceso de obtención de la solución se evitan ejecutando correctamente el algoritmo. Con todo esto, son esas propiedades especiales de las matrices de elementos finitos, no

compartidas con los métodos de ecuaciones integrales, lo que hace al método de elementos finitos más atractivo para el análisis de problemas electromagnéticos a gran escala.

El segundo problema se soluciona habitualmente con soluciones *hardware* con el uso de grandes equipos o de *clusters* (un conjunto de equipos que trabajan simultáneamente de forma coordinada) o con soluciones *software* de optimización de códigos o paralelizacion de los mismos.

2.5.1 Código HOFEM

En el grupo de investigación al que pertenecen el tutor y el director del trabajo fin de grado se ha desarrollado una línea de investigación basada en la simulación electromagnética y, en concreto, en el Método de los Elementos Finitos.

Uno de los resultados de esta línea de investigación es el desarrollo de una *suite* de simulación electromagnética llamada HOFEM [5], de la que se pueden distinguir dos partes fundamentales:

- Interfaz gráfica, disponible en Windows y Linux usando el lenguaje *TCL-TK* [20] y basándose en la herramienta GiD [21], que se encarga de hacer el pre-proceso y el post-proceso del problema.
- Núcleo, o *core*, disponible también en Windows y Linux y que usa el lenguaje *FORTRAN* para obtener la solución del problema electromagnético que se desea simular. La interacción con la interfaz se basa en el paso al núcleo de unos archivos de entrada, con los que se generan unos archivos de salida que serán devueltos a la interfaz para que se encargue del post-proceso.

Las principales características de este simulador son:

- Uso de construcciones modernas de FORTRAN con OOP (*Object Oriented Programming*).
- Posibilidad de hacer simulación secuencial y paralela por medio de MPI (*MessagePassing Interface* [22]).
- *Solvers in-core* (usando únicamente memoria virtual) y *out-of-core*(usando directamente el sistema de disco para almacenar las matrices) para solucionar el sistema de ecuaciones.
- Diferentes elementos de discretización del dominio: tetraedros y prismas de base triangular.
- Posibilidad de utilizar remotamente *clusters* de altas prestaciones de una forma intuitiva.

2.6 SOLVERS

Existen dos estrategias que se pueden utilizar para resolver un sistema de ecuaciones.

- **Métodos directos:** este método permite resolver sistemas de ecuaciones utilizando técnicas numéricas exactas para invertir matrices, basadas en la factorización LU, eliminación por Gauss-Jordan, etc.

- **Métodos iterativos:** en este caso, el método utiliza técnicas de aproximación para encontrar una solución suficientemente buena y repite el proceso hasta que el cambio entre un paso y otro no varíe significativamente o hasta que se cumpla un criterio de error (y por lo tanto el proceso iterativo converja). La diferencia principal con un método directo radica en que en los métodos iterativos se puede suspender el proceso al término de una iteración y se obtiene una aproximación a la solución. Puesto que lo que se calcula con un método de estas características es una *aproximación* a la solución, los métodos iterativos se utilizan cuando no se conoce un método para obtener la solución de forma exacta o bien, cuando el método para determinar la solución exacta requiere mucho tiempo de cálculo, una respuesta aproximada es adecuada.

2.6.1 Solvers Directos

Estos *solvers* utilizan las ecuaciones y las resuelven directamente (en un solo paso). Un ejemplo típico es la utilización del método de la matriz inversa:

$$A \cdot X = B$$
$$A^{-1} \cdot A \cdot X = A^{-1} \cdot B$$
$$X = A^{-1} \cdot B$$

Puesto que en el análisis de elementos finitos el número de ecuaciones es muy alto, calcular la inversa de la matriz supone un consumo de recursos importante. La característica más importante de este tipo de *solvers* es que, resuelve un grupo de ecuaciones de forma directa, es decir, sin ningún tipo de aproximación y, por tanto, no existen errores asociados al proceso de resolución. El principal problema radica en que las matrices de un método de elementos finitos (FEM) son dispersas y al hacer la inversa se convierten en matrices densas (ocupando mucho espacio de almacenamiento). De esta forma, es habitual la resolución de este tipo de sistemas utilizada el algoritmo LU (*lower-upper*), que permite no calcular la inversa de la matriz.

LU

LU es una técnica para la resolución de sistemas de ecuaciones muy popularizada. Muchas librerías para la solución de estos sistemas, por ejemplo MUMPS, de la que hablaremos a continuación, están basadas en esta técnica.

Es una forma de factorización de una matriz como el producto de una matriz triangular inferior y una superior. Esta descomposición se usa en el análisis numérico para resolver sistemas de ecuaciones (más eficientemente) o encontrar matrices inversas.

$$A = LU$$

Pasos para resolver un sistema de ecuaciones por el método de descomposición LU:

- Obtener la matriz triangular inferior 'L' y la matriz superior 'U'.
- Resolver $Ly = b$ (para encontrar y).
- El resultado del paso anterior se guarda en un vector nuevo de nombre 'y'.
- Realizar $Ux = y$(para encontrar x).
- El resultado del paso anterior se almacena en un vector nuevo llamado 'x', que incluye los valores correspondientes a las incógnitas de la ecuación.

A continuación se expone el funcionamiento de dos de los *solvers* directos más utilizados, MUMPS (*Multifrontal Massively Parallel Sparse Direct Solver*) y SuperLU.

MUMPS

MUMPS es una librería para la solución de grandes sistemas de ecuaciones algebraicas lineales dispersas en computadores de memoria paralela. Se trata de una versión optimizada de la técnica LU para problemas específicos de matrices densas. De esta forma implementa un método multi-frontal, que es una versión de la eliminación de Gauss para la resolución de sistemas de ecuaciones lineales dispersos. La importancia de MUMPS radica en el hecho de que es una implementación más conocida del método multi-frontal.

MUMPS resuelve $Ax = b$ en tres pasos principales:

1. Análisis
    a. Ordenación.
    b. Factorización simbólica.
2. Factorización $A = LU$
    a. A se distribuye a los procesadores.
    b. La factorización numérica en cada matriz es realizada por un procesador 'maestro' y uno o más procesadores 'esclavos'.
3. Solución
    a. b se transmite desde el host.

b. x se calcula utilizando factores distribuidos.
c. La solución puede montarse en el host o mantenerse distribuido en los procesadores.

SUPERLU

SuperLU es una librería de uso general para las soluciones directas de grandes sistemas, dispersos y no simétricas de ecuaciones lineales. Esta librería está escrita en C y puede ser utilizada tanto desde C como desde FORTRAN.

- Las rutinas contenidas en este paquete realizan una descomposición LU con pivote parcial, y resuelve el sistema triangular mediante la sustitución al principio y al final.
- Las rutinas de factorización LU pueden manejar matrices no cuadradas, pero las soluciones triangulares se realizan sólo para matrices cuadradas.
- Las columnas de la matriz puedes estar pre-ordenadas (antes de la factorización), ya sea a través de la propia librería o de rutinas suministradas por el usuario.

2.6.2 Solvers Iterativos

Estos *solvers* utilizan técnicas de aproximación para obtener una solución, es decir, se aproximan ciertos parámetros con la intención de resolver las ecuaciones y posteriormente se calculan los errores asociados. El algoritmo evalúa las ecuaciones matriciales para determinar la bondad de la predicción, ajustando el siguiente paso en función del error de cálculo. Las iteraciones continúan hasta que el error llega a un valor suficientemente pequeño ya que, a medida que va convergiendo el resultado se acerca a la solución correcta. El programa define unos límites de error que establecen cuando el cambio entre paso y paso es suficientemente pequeño y, por tanto, es posible parar el cálculo.

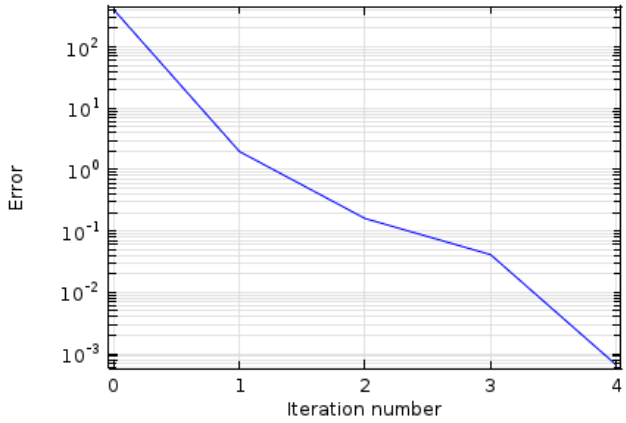


Figura 10. Gráfica de convergencia para un método iterativo.

CAPÍTULO 2: ENTORNO DE TRABAJO

A continuación, se detalla el funcionamiento de los *solvers* iterativos utilizados durante el desarrollo del presente proyecto.

JACOBI

El método de Jacobi es un método iterativo usado para resolver sistemas de ecuaciones lineales del tipo $\boldsymbol{Ax} = \boldsymbol{b}$. El algoritmo toma su nombre del matemático alemán Carl Gustav Jakob Jacobi.

Este método consiste en realizar una secuencia de transformaciones ortogonales, cada transformación se denomina 'rotación de Jacobi', y corresponde a un paso cuyo objetivo es eliminar un elemento de la matriz. Se va rotando sucesivamente la matriz hasta que el error es lo suficientemente pequeño para ser considerada una matriz diagonal.

Para determinar si el método de Jacobi converge hacia una solución, se evalúa la siguiente condición de convergencia:

1. Si la matriz de coeficientes original del sistema de ecuaciones es diagonalmente dominante, es decir, si el elemento de la diagonal correspondiente a la fila *i* es mayor que la suma de los elementos de esa fila *i*, el método de Jacobi converge.

La secuencia se construye descomponiendo la matriz del sistema $\boldsymbol{A}$ en la forma siguiente: $\boldsymbol{A} = \boldsymbol{D} + \boldsymbol{L} + \boldsymbol{U}$ siendo $\boldsymbol{D}$ una matriz diagonal, $\boldsymbol{L}$ una matriz triangular inferior y $\boldsymbol{U}$ una matriz triangular superior. Partiendo de $\boldsymbol{Ax} = \boldsymbol{b}$, podemos reescribir la ecuación anterior como: $\boldsymbol{Dx} + (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x} = \boldsymbol{b}$

Conjugate Gradient (CG)

El método del Gradiente Conjugado es un algoritmo para solución de sistemas de ecuaciones lineales en los que la matriz es simétrica y definida positiva. Se trata de un método iterativo, aplicable a sistemas dispersos que son demasiado grandes para la aplicación de métodos directos.

Este método se aplica a la ecuación $\boldsymbol{A} \cdot \boldsymbol{u} = \boldsymbol{b}$, cuando $\boldsymbol{A}$ es definida positiva y simétrica. El funcionamiento de este método consiste en construir una base de vectores ortogonales y utilizarla para realizar la búsqueda de la solución de la forma más eficiente. La gran ventaja del método del Gradiente Conjugado radica en que basta con asegurar la ortogonalidad de un nuevo elemento con respecto al último que se ha generado, para que automáticamente se cumpla esta condición con respecto a todos los anteriores.

Entre los distintos métodos iterativos disponibles para obtener la solución de un sistema de ecuaciones, el método del conjugado gradiente merece más atención debido a que, en principio, proporciona una solución exacta cuando el número de iteraciones alcanza el número de ecuaciones o incógnitas.

Por último, merece la pena destacar que, en la mayoría de los casos, los *solvers* directos utilizarán mayor cantidad de memoria que los *solvers* iterativos, pero serán más robustos. Los métodos iterativos se aproximarán a la solución gradualmente, y es posible cambiar la tolerancia de convergencia si así se desea.

Capítulo 3

Metodología de Trabajo

El código FORTRAN desarrollado en la Universidad Carlos III de Madrid, HOFEM para el tratamiento de problemas de Elementos Finitos contiene diversas características que son imprescindibles para su funcionamiento, y que por tanto, es necesario adaptar a la interfaz de Python:

1. Separación clara de ficheros, archivos *program* que únicamente contienen llamadas a las subrutinas y los ficheros donde se almacenan dichas subrutinas.
2. Uso de *allocatable* arrays, arrays en los que no se especifica su tamaño al declararlos.
3. Uso de tipos REAL (KIND), declaración explícita del tipo de las variables.
4. Interconexión con las librerías de HOFEM.

En este capítulo, se exponen dichas características, así como los problemas encontrados durante el proceso de transformación de FORTRAN a Python.

3.1 Paso0 – Funcionamiento de F2PY

Durante esta primera fase, el objetivo principal fue comprender el funcionamiento interno de F2PY, que incluía la utilización de bibliotecas escritas en FORTRAN desde Python, la posibilidad de preparar el código FORTRAN para que este proceso fuera lo más sencillo posible y cómo solventar los primeros problemas que pudieran surgir.

El primer código FORTRAN que se trató de encapsular fue el clásico '*Hola Mundo*'. Como se ha descrito anteriormente, para poder controlar el flujo desde Python, es necesario transformar el código FORTRAN en una subrutina o bien en un módulo, posteriormente se compila mediante f2py creando un archivo .py (extensión de los módulos de Python) y, una vez logrado esto, podemos invocar las funciones contenidas en este módulo desde el intérprete de Python mediante la opción *'import'.*

A continuación se expone el funcionamiento de F2PY mediante el citado código de 'Hola Mundo':

```
print *, 'Hola, mundo!'
end
```

Para poder utilizar el ejemplo anterior a través de la interfaz de Python, es necesario convertir el código FORTRAN en una subrutina/módulo:

```
! hola_mundo_sub.f90
Subroutine hola_mundo(msg)
    Character(len=12), intent(out) :: msg
    msg = 'Hola, mundo!'
end subroutine
```

En este punto ya es posible invocar F2PY mediante el siguiente comando:

```
$ f2py –c hola_mundo_sub.f90 –m hola_mundo_sub
```

Con el argumento –c se indica el fichero a compilar, y con –m el nombre del módulo Python resultante. Esta línea crea un archivo llamado 'hola_mundo_sub.so'. Desde el intérprete de Python se puede acceder a este nuevo módulo mediante la opción 'import':

```
In [1]: !ls
hola_mundo.f90  hola_mundo_sub.f90  hola_mundo_submodule.c  hola_mundo_sub.so
In [2]: import hola_mundo_sub
In [3]: hola_mundo_sub?  # Documentación automática del módulo
Type:        module
String Form:<module 'hola_mundo_sub' from 'hola_mundo_sub.so'>
```

```
Docstring:
This module 'hola_mundo_sub' is auto-generated with f2py (version:2).
Functions:
msg = hola_mundo()

In [4]: hola_mundo_sub.hola_mundo?  # Documentación automática de la subrutina
Type:        fortran
String Form:
Docstring:
hola_mundo - Function signature:
msg = hola_mundo()
Return objects:
msg : string(len=12)
In [5]: hola_mundo_sub.hola_mundo()  # ¡Funciona!
Out[5]: 'Hola, mundo!'
```

Figura 11. Código 'Hola Mundo' en Python

F2PY transforma los argumentos de la función. El código FORTRAN constaba de una subrutina con el argumento 'msg' únicamente de salida con 'intent(out)' y F2PY lo ha transformado en un valor de retorno de la función en Python.

3.2 Paso1 – Llamadas a módulos

Python puede colocar definiciones en un archivo y utilizarlo como script, o en una instancia interactiva del intérprete. Tal archivo es llamado módulo y las definiciones de un módulo pueden ser importadas a otros módulos o al módulo principal.

Un módulo es un archivo contenedor de definiciones y declaraciones en Python. El nombre del módulo es el nombre del archivo con el sufijo *.py* agregado. Es importante destacar que el nombre del módulo no puede contener caracteres no reconocidos por Python, sino al realizar la importación, devolverá el siguiente error:

'SyntaxError:Non-ASCIIcharacter'

Del mismo modo, es necesario señalar el hecho de que FORTRAN no es '*case sensitive'*, pero Python sí, lo que puede derivar en conflictos al nombrar los módulos. Durante la realización de estas pruebas, se encontraron diversos problemas derivados de esta casuística, ya que para acceder a las funcionalidades de un módulo de Python creado es necesario escribir el nombre de este en minúsculas al realizar la importación, algo a lo que no se hace referencia en la documentación de f2py.Por ejemplo, si se compila el siguiente módulo mediante F2PY:

$ f2py –c modulo_sub.f90 –m Modulo_sub

Para acceder a sus funciones desde la interfaz de Python, será de la siguiente manera:

$ Import modulo_sub

Una vez conseguido este primer objetivo, se construyó un nuevo programa en FORTRAN desde cero, en concreto, una subrutina que mostrara *'n'* asteriscos por pantalla, y un fichero '*program'* que la invocara. Es importante destacar el hecho de que la subrutina, y el programa que hacía uso de ella se encontraba en ficheros separados, algo necesario debido a cómo se encontraba dispuesto el código FORTRAN para tratar con problemas de Elementos Finitos, dónde hay una separación clara de ficheros. Se consiguió separar dicho programa en dos ficheros independientes y compilar ambos mediante f2py, resultando en un único módulo accesible desde Python.

Se incluye a continuación, el código de este ejemplo y su ejecución en Python:

```
! asteriscos.f90
Subroutine asteriscos (n)
implicit none

integer n
integer i

do i =1, n
print *, '*'
end do
end subroutine
```

```
!programa.f90
program programa
implicit none

call asteriscos (n)
end
```

Compilamos ambos ficheros mediante f2py, es importante señalar que el programa principal debe anteponerse al resto de subrutinas.

$ f2py –c programa.f90 asteriscos.f90 –m asteriscos

```
C:\Users\Carlos\Anaconda\TFG\PRUEBAS>python
Python 2.7.8 |Anaconda 2.1.0 (64-bit)| (default, Jul  2 2014, 15:12:11) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://binstar.org
>>> import asteriscos
>>> asteriscos.asteriscos(5)
 *
 *
 *
 *
 *
```

Figura 12. Ejecución del Módulo 'Asteriscos'

Ya que el objetivo era poder utilizar f2py con un problema de Elementos Finitos (FEM), se construyó un nuevo módulo: '*vectores.f90*' que incluía 2 funciones para trabajar con arrays: cálculo del producto vectorial y producto escalar a partir de dos arrays pasados como parámetro:

Como se ha explicado con anterioridad, el código siguiente puede compilarse mediante f2py, resultando en un módulo accesible desde la interfaz de Python, pudiendo invocar las funciones contenidas en él.

```
module vectores

   implicit none

   contains

   ! Producto escalar entre dos vectores u, v de longitud n
   function producto_escalar(n, u, v) result(p)

      integer, intent(in) :: n
      double precision, intent(in) :: u(n), v(n)
      double precision :: p

      p = dot_product(u, v)

   end function

   ! Producto vectorial entre dos vectores u, v de longitud 3
   function producto_vectorial(u, v) result(w)

      double precision, intent(in) :: u(3), v(3)
      double precision :: w(3)

      w(1) = u(2) * v(3) - u(3) * v(2)
      w(2) = u(3) * v(1) - u(1) * v(3)
      w(3) = u(1) * v(2) - u(2) * v(1)

   end function

end module
```

Figura 13. Código FORTRAN del módulo Vectores

Se dividió el anterior código 'vectores' en 3 ficheros independientes, uno para cada función y otro para el programa principal que realizaría las llamadas a ambas funciones. Compilando todos los archivos juntos con F2PY en un único módulo de Python, resultando en las mismas funcionalidades que tratándose de un único fichero, un requisito imprescindible para poder abordar posteriormente un código de mayor envergadura.

$ f2py –c vectores.f90 producto_escalar.f90 producto_vectorial.f90 –m vectores

```
Python 2.7.8 |Anaconda 2.1.0 (64-bit)| (default, Jul  2 2014, 15:12:11) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://binstar.org
>>> from vectores import *
>>> import numpy as np
>>> u = np.array([1,2,3])
>>> v = np.array ([1,0,-1])
>>> vectores.producto_escalar(u,v)
-2.0
>>> vectores.producto_vectorial(u,v)
array([-2.,  4., -2.])
```

Figura 14. Ejecución del Módulo 'Vectores'

3.3 Paso2 – Uso de tipos REAL (KIND)

Durante el proceso de encapsulado del código FORTRAN, surgieron complicaciones debido a la utilización de tipos REAL (KIND=DBL) de forma recurrente durante todo el programa. F2PY no era capaz de reconocer este tipo de argumentos, y por tanto, imposibilitaba la creación de los módulos *.py*. Se descubrió la necesidad de utilizar*f2cmap* para adaptar estos tipos a la interfaz de Python.

En la tabla siguiente se define como se asignan los diferentes tipos de variables en FORTRAN a tipos C compatibles con f2py. Los usuarios pueden redefinir estas asignaciones mediante la creación de un archivo *f2cmap.f2py* [21] en el directorio de trabajo. Este archivo debe contener un diccionario de Python, por ejemplo, *{'real':{'low':'float'}}* que informa a f2py para asignar el tipo FORTRAN real *(low)* al tipo *float*de C, o de forma más general, el fichero *f2cmap.f2py* debe contener una sentencia de este tipo:

<Fortran typespec> : {<selector_expr> : <C type>}

Que define la asignación entre el tipo FORTRAN:

<Fortran typespec> : ([kind=]<selector_expr>)

Y el correspondiente tipo de C, más adelante se incluye una tabla con las posibles asignaciones de tipos FORTRAN a tipos C.

En este caso, la conversión incluida en el fichero *f2cmap* es: *{'REAL':{'DBL':'complex_double'}}.*

Fortran type	C type
integer *1	char
byte	char
integer *2	short
integer[| *4]	int
integer *8	long long
logical *1	char
logical *2	short
logical[| *4]	int
logical *8	int
real[| *4]	float
real *8	double
real *16	long double
complex[| *8]	struct {float r,i;}
complex *16	struct {double r,i;}
complex *32	struct {long double r,i;}
character[*...]	char *

Tabla 3. Asignación de tipos Fortran a tipos C.

Para poder adaptarlo satisfactoriamente al código, se decidió el cambio por un tipo *'complex_double'*.

Como se ha comentado anteriormente, debido a que Python es *case sensitive*, al construir el diccionario para la conversión de los tipos REAL(kind=DBL), es necesario incluir todas las combinaciones posibles de mayúsculas y minúsculas que puedan darse en FORTRAN, y que Python sería capaz de diferenciar.

```
C:\Users\Carlos\Anaconda\TFG\PRUEBAS\PRUEBA F2CMAP>f2py -c vectores.f90 -m vectores
Reading .f2py_f2cmap ...
        Mapping "REAL(kind=DBL)" to "complex_double"
        Mapping "REAL(kind=dbl)" to "complex_double"
        Mapping "real(kind=DBL)" to "complex_double"
        Mapping "real(kind=dbl)" to "complex_double"
Succesfully applied user defined changes from .f2py_f2cmap
running build
running config_cc
unifing config_cc, config, build_clib, build_ext, build commands --compiler options
running config_fc
```

Figura 15. Uso de f2cmap

Es importante señalar que, una vez comprobado el buen funcionamiento de *f2cmap*, si se utiliza en un módulo de Python, por ejemplo, con el módulo '*vectores*'mencionado anteriormente, es necesario importar el módulo *numpy* (no existe referencia al respecto en la documentación de *f2cmap*)ya que f2cmap está incluido, como la mayoría de funciones externas de Python, dentro del módulo *numpy.*

```
C:\Users\Carlos\Anaconda\TFG\PRUEBAS\PRUEBA F2CMAP>python
Python 2.7.8 |Anaconda 2.1.0 (64-bit)| (default, Jul  2 2014, 15:12:11) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://binstar.org
>>> from vectores import *
>>> import numpy as np
>>> u = np.array([1, 2, 3])
>>> v = np.array ([1, 0, -1])
>>> vectores.producto_escalar(u,v)
-2.0
>>>
```

Figura 16. Uso de f2cmap (II)

3.4 Paso3 – Uso de allocatable arrays

En varios puntos de la documentación de f2py, se advierte de sus limitaciones con respecto al encapsulado de características 'modernas' de FORTRAN. Para el presente proyecto, es fundamental el poder manejar adecuadamente *allocatable arrays,* introducidos en la especificación de FORTRAN 90.

De esta forma, se diseñó un código simple para comprobar las prestaciones de f2py con respecto a este tipo de arrays. Dicho módulo se compiló correctamente con *f2py*, siendo accesible desde la interfaz de Python a través de la opción '*import*'.

```
! alloc_test.f90
subroutine f(x, z)
  implicit none

! Argument Declarations !
  real*8, intent(in) ::  x(:)
  real*8, intent(out) :: z(:)

! Variable Declarations !
  real*8, allocatable :: y(:)
  integer :: n

! Variable Initializations !
  n = size(x)
  allocate(y(n))

! Statements !
  y(:) = 1.0
  z = x + y

  deallocate(y)
  return
end subroutine f
```

Figura 17. Código FORTRAN del módulo 'alloc'

Sin embargo, al probar las funciones contenidas en este módulo, se obtenía el siguiente error:

```
>>> from alloc import f
>>> from numpy import ones
>>> x = ones(5)
>>> print f(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: failed to create intent(cache|hide)|optional array-- must have defined dimensions but got (-1,)
>>>
```

Figura 18. Error allocatable arrays.

Se siguieron algunas recomendaciones acerca de la escritura de código FORTRAN para su posterior uso desde una interfaz de Python [22], resultando en una variación del código anterior. Con este nuevo programa, el tamaño de los arrays 'x' e 'y' se pasa a la función como argumento explícito, y por lo tanto, *f2py* hace que dicho argumento 'n' sea opcional. A continuación se muestra la documentación de dicha función tal y como aparece en la interfaz de Python.

```
Type:       fortran
String Form:<fortran object>
Docstring:
f - Function signature:
  z = f(x,[n])
Required arguments:
  x : input rank-1 array('d') with bounds (n)
Optional arguments:
  n := len(x) input int
Return objects:
  z : rank-1 array('d') with bounds (n)
```

Figura 19. Documentación de la función sobre arrays allocatables en Python

F2py interpreta correctamente que el argumento 'n' es el tamaño de los arrays, y lo convierte en un parámetro opcional. A la hora de manejar arrays de esta manera, se dice que se dan los arrays de 'forma explícita'; existe otra manera, denominada de 'forma asumida'. Un array de 'forma asumida' es un array de una subrutina que no es local y que tiene un determinado tipo y rango. La extensión de dicho array se determina a través de una interfaz explícita en el código del fichero *program* que realiza la llamada a la subrutina, de modo que la forma de un array de estas características no es conocida, sino que toma cualquier forma dependiendo de un parámetro actual. El uso de arrays de este tipo resulta en diversos problemas con *f2py* y por tanto se desaconseja su uso.

```
C:\Users\Carlos\Anaconda\TFG\allocate>python
Python 2.7.8 |Anaconda 2.1.0 (64-bit)| (default, Jul  2 2014, 15:12:11) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://binstar.org
>>> from alloc1 import f
>>> from numpy import ones
>>> x = ones(5)
>>> print f(x)
[ 2.  2.  2.  2.  2.]
```

Figura 20. Ejecución del módulo sobre arrays allocatables en Python.

3.5 Paso4 – Interconexión con librerías de HOFEM

Este módulo tratará sobre las pruebas finales realizadas con el código FORTRAN (modificado para su encapsulación y uso a través de la interfaz de Python) y su compilación con las librerías de HOFEM.

Debido a los resultados obtenidos con pruebas anteriores, se decidió dividir el código en varios ficheros, teniendo un fichero '*program'* que contendría únicamente las llamadas al resto de funciones, de manera similar a las pruebas que se realizaron con el módulo '*vectores'.* Este programa principal se encuentra en el fichero '*FEMPython.F90*' y utiliza un driver (fichero que llama a rutinas de HOFEM) llamado '*MainDriversPy.F90*' y que contiene las siguientes subrutinas:

! Inicializar los parametros del programa.

SUBROUTINE initiate_program_parameters

! Leer el fichero de entrada.

SUBROUTINE readInputData

! Inicializa el entorno de trabajo de HOFEM y obtiene el número de elementos no nulos de la matriz necesaria para resolver el problema electromagnético. nnz_mumps es el parámetro de salida.

SUBROUTINE py_initialize_system(nnz_mumps)

! Se encarga de ensamblar las matrices que van a ser resueltas por MUMPS en el formato coo y obtener el right-hand side para resolver el problema. Todos los parámetros son de salida.

SUBROUTINE py_assembly_system_equations(A_loc, RHS, IRN_loc, JCN_loc)

! Se encarga de hacer el post-proceso con la solución obtenida con MUMPS o con el resolvedor de Python.

SUBROUTINE py_postprocess_write_gi_coefs(FEMsolution)

! Cierra HOFEM cerrando el parallel environment y los archivos de disco.
SUBROUTINE py_finalize()

La compilación de este programa con *f2py* se realizaba correctamente, pero, al intentar acceder a ese módulo desde la interfaz Python mediante la opción '*import'* se obtenía el siguiente error.

*ImportError: ./program.so: undefined symbol: ran2_*

Se realizó una búsqueda al respecto, en la que se descubrió que pueden existir diversos problemas con *f2py* si el código FORTRAN original contiene llamadas a funciones externas, para este caso concreto, el error de importación era dado por esta llamada perteneciente a la librería de MPI (*MessagePassing Interface*), indispensable en HOFEM.

Syscontrolparameters_mp_mpirank.

Al ser la misma casuística, se obtiene este error debido a que es la primera función que *f2py* no es capaz de reconocer, omitiendo esta función, el error persistía con la siguiente llamada irreconocible.

Otro posible problema con ese programa, vendría dado por un *linkado* de librerías, según la documentación de *f2py*, se trataría de un problema común que puede solucionarse utilizando *SWIG* (añadiendo links de librerías a '*extra_compile_args*').SWIG es una herramienta de desarrollo de software que conecta programas escritos en C, C++ y FORTRAN con una gran variedad de lenguajes de programación de alto nivel.

Llegados a este punto, eran varios los errores de compilación obtenidos:

Uno de dicho errores era debido a diversas librerías 'conflictivas'. Si se prueba a descartar dichas librerías, *f2py* era capaz de compilar el programa correctamente, sin embargo, no generaba el archivo *.py (*extensión de los módulos de Python*),* sino que únicamente mostraba el archivo *.so* (complementario al anterior), lo cual se reflejaba en el siguiente error al realizar la importación.

*ImportError: ./upper_main_driver_py.so: undefined symbol: gid_beginscalarresult_*

El problema principal sigue viniendo dado por el *linkado* de algunas librerías, éstas, en teoría, no conflictivas, y de requerimiento básico para nuestro código. Es posible que debido a fallos en la arquitectura de estas librerías externas, se obtenía el siguiente error al realizar la compilación con *f2py.*

Libmkl_lapack95_lp64.a: could not read symbols: Bad value

CAPÍTULO 3: METODOLOGÍA DE TRABAJO

Llegados a este punto, no fue posible continuar con el desarrollo del proyecto, ya que existe poca documentación al respecto, y dada la posibilidad de que determinadas librerías imprescindibles, no fueran compatibles con *f2py*, como se advierte en la documentación, se optó por dejar a un lado esta vía de acción, y centrarse en la resolución de matrices obtenidas desde HOFEM a través de escritura en disco, sin recorrer el paso intermedio de encapsular el código FORTRAN mediante *f2py.*

Capítulo 4

Estudio comparativo entre Solvers.

Se han obtenido las matrices que corresponden a la simulación electromagnética de una guía de onda funcionando a una frecuencia de trabajo de 6 Ghz. El tamaño de la matriz es más grande conforme es más fino el mallado de la guía; de esta forma se obtienen 6 tamaños diferentes de matriz para hacer un estudio comparativo de las prestaciones de los diferentes solvers.

Matriz	Tamaño_Matriz	Tamaño_IRN	Max_IRN	Tamaño_RHS
A_loc0	3003	3003	194	388
A_loc1	4635	4635	258	516
A_loc2	5484	5484	296	592
A_loc3	9705	9705	518	1036
A_loc4	28989	28989	1486	2972
A_loc5	82710	82710	4084	8168

Tabla 3. Matrices HOFEM.

Es importante señalar que el vector RHS se obtiene de las condiciones de contorno del problema electromagnético analizado, y que las matrices proporcionadas por HOFEM se encuentran en Formato Coordinado (COO) [25], se trata de un formato de uso muy extendido cuyo funcionamiento se detalla a continuación.

Los vectores: valores, fila y columna almacenan el valor de los elementos no nulos de la matriz, los índices de la fila y los índices de la columna respectivamente. COO es una representación general de una matriz dispersa, donde la capacidad de almacenamiento requerida depende de los valores no nulos. Seguidamente, se expone un ejemplo de su utilización con una matriz $2x2$.

$$A = \begin{matrix} 5 & 3 \\ 1 & 2 \end{matrix}$$

1. Valores $[5, 3, 1, 2]$
2. Filas $[1, 1, 2, 2]$
3. Columnas $[1, 2, 1, 2]$

Este formato es muy eficiente si se quiere acceder de forma secuencial a todos los elementos, como por ejemplo, si se quisiera realizar el producto $matriz\ x\ vector$,la velocidad de algoritmo usado para ese programa sería más rápida y eficiente que si se utilizaran otro formato.

Se decidió utilizar la implementación de PETSc para Python, Petsc4Py (de la que se ha hablado anteriormente), debido a la posibilidad de utilizar diferentes *solvers* a través de su interfaz. Dado que el tratamiento de vectores y matrices no se da de la misma manera en Petsc4Py y en Python, fue necesario realizar pruebas iniciales para comprender el funcionamiento de esta herramienta, de forma similar a como se hizo con F2Py.

4.1 Paso0 – Funcionamiento de Petsc4Py

De nuevo, al igual que ocurría con F2Py, la documentación sobre esta herramienta es cuanto menos escasa, si bien, existe gran cantidad de documentación y ejemplos acerca de PETSc [15], por lo que es posible adaptar gran parte del código de dichos ejemplos a una interfaz de Python.

En el sitio oficial de PETSc [15] podemos encontrar dicha documentación y ejemplos, así como una tabla en la que se especifican los *solvers* disponibles en esta suite. Para cada una de las funciones que contiene PETSc (y que pueden ser replicadas en Petsc4Py), hay documentación disponible y ejemplos al respecto, por ejemplo, la función *MatCreate()* (imprescindible para la resolución de cualquier sistema de ecuaciones en PETSc), tiene su propia página de documentación [26] en la que se describe su funcionamiento, así como enlaces a funciones relacionadas:

Creates a matrix where the type is determined from either a call to MatSetType() or from the options database with a call to MatSetFromOptions(). The default matrix type is AIJ, using the routines MatCreateSeqAIJ() or MatCreateAIJ() If you do not set a type in the options database. If you never call MatSetType() or MatSetFromOptions() it will generate an error when you try to use the matrix.

A partir de uno de dichos ejemplos de PETSc, se construyó un código en Python para la resolución de sistemas de ecuaciones con matrices generadas de forma aleatoria. A continuación se incluye el detalle de la ejecución de dicho código, generando una matriz aleatoria de tamaño 1000, y donde se muestra la solución obtenida para dos de los *solvers* que se emplearan posteriormente con las matrices de Elementos Finitos: Conjugado Gradiente y GMRES.

```
[cromero@ash25v Scripts]$ python prueba.py
Ingrese el numero de filas:1000
Ingrese el numero de columnas:1000
La solucion del sistema es:
0.000281209050312
0.000511395844024
0.000710733912595
0.000888713895302
0.00105051826852
0.00119935015067
0.0013373663965
0.00146610042689
0.0015867211676
```

Figura 21. Solución mediante el método del Conjugado Gradiente

```
[cromero@ash25v Scripts]$ python prueba.py
Ingrese el numero de filas:1000
Ingrese el numero de columnas:1000
La solucion del sistema es:
0.000281203160773
0.0005113917539
0.000710731486819
0.000888707332234
0.00105051036464
0.00119934313805
0.00133735906867
0.00146609594956
```

Figura 22. Solución mediante el método GMRES

A continuación se detalla en varios formatos los tiempos de resolución de los solvers que se emplearán posteriormente para el problema de Elementos Finitos; Conjugado Gradiente, GMRES, Jacobi, KSP, LU y SOR, aplicados a la matriz aleatoria de tamaño 1000 mencionada anteriormente.

	CG	GMRES	JACOBI	KSP	LU	SOR
A_prueba	0,0147852	0,0167452	0,0154728	0,0217481	0,018487	0,016874

Tabla 4.Tiempo de resolución para distintos solvers.

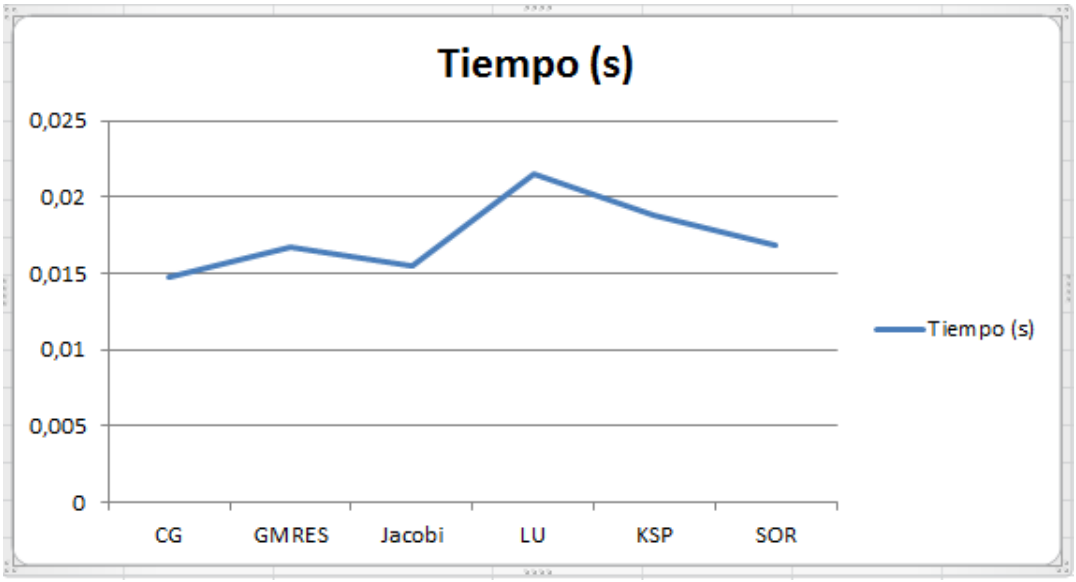


Figura 23. Tiempos de Ejecución para Distintos Solvers

A partir de este código, se aprecia el funcionamiento de Petsc4Py, y las diferencias con Python a la hora de tratar vectores y matrices, si bien es posible combinar ambos en

el mismo código, solo las matrices y vectores generadas por Petsc4Py pueden utilizarse para resolver un sistema de ecuaciones.

Es importante destacar que, en la página oficial de PETSc se incluye una fe de erratas donde están contemplados todos los cambios llevados a cabo durante el cambio de versión. Durante las primeras pruebas, no se conseguía resolver ningún sistema debido a la aparición de este error recurrente:

[0] Object is in wrong state
[0] Must call MatXXXSetPreallocation() or MatSetUp() on argument 1 "mat" before MatCreateVecs

En los cambios correspondientes a la versión 3.3 se especifica que es necesario el uso de una de las dos funciones que se detallan en el error antes de generar vectores y/o matrices de PETSc, sin embargo, al intentar invocar a alguna de estas funciones mencionadas, se obtenía el siguiente error:

AttributeError: 'petsc4py.PETSc.Mat' object has no attribute 'MatSetUp'

De nuevo un error asociado al *case sensitive* de Python, ya que está función viene definida en la documentación de PETSc como: MatSetUp() mientras que en Python es MatsetUp().

Debido a la ausencia de documentación respecto a la utilización de las funcionalidades de PETSc a través de la interfaz de Python, el uso de la opción *help* de Python ha sido imprescindible para la comprensión y utilización de Petsc4Py, dadas las características de dicha función de ayuda, es posible obtener información sobre todas las funciones relacionadas con matrices y/o vectores disponibles en PETSc.

$ import petsc4py
$ from petsc4py import PETSc
$ help (petsc4py.PETSc.Mat)
$ help (petsc4py.PETSc.Vec)

4.1 Paso1 – Tratamiento de las matrices HOFEM desde Python

Dado que las matrices proporcionadas por HOFEM estaban en formato .txt y al tamaño de alguna de ellas, se construyó un código en Python para tratar con dichas matrices. Dicho código, leía línea a línea el fichero .txt y guardaba todos los elementos en un array construido previamente.

Existen diversas funciones contenidas dentro de NumPy que permiten la creación de matrices en Python con la ventaja añadida de que no es necesario preocuparse por el manejo de ficheros con Python, ya que leer o escribir desde/en un fichero se reduce a invocar una función.

NumPy incorpora varias funciones para cargar datos de forma matricial, la más utilizada es *loadtxt(),* que consta de un único argumento obligatorio, el nombre del archivo o un objeto *file* desde el que leer los datos.

Para dicha función es posible incorporar argumentos opcionales para leer todo el fichero o ignorar ciertas líneas, por ejemplo, aquellas que empiecen por # (comentarios en Python). El delimitador que marca la separación de los datos (la función espera una separación mediante espacios) y el tipo de array resultante (por defecto, de tipo *float*).

Para este caso, debido a las características propias de las matrices generadas por HOFEM, se decidió utilizar la función *open()* y posteriormente guardar esas matrices en arrays de Numpy.

```
aux1 = open ('IRN_loc5.txt', 'r')
aux2 = open ('JCN_loc5.txt', 'r')
aux3 = open ('A_loc5.txt', 'r')
aux4 = open ('RHS_loc5.txt','r')

anp.IRN = []
anp.JCN = []
anp.A = []
anp.RHS = []

for i in aux1: # read rest of lines
    anp.IRN.append([int(i)])

for i in aux2: # read rest of lines
    anp.JCN.append([int(i)])

for i in aux3: # read rest of lines
    anp.A.append([complex(i)])

for i in aux4: # read rest of lines
    anp.RHS.append([complex(i)])
```

Figura 24. Código Python para leer las matrices de HOFEM

Una vez transformadas las matrices de HOFEM a arrays de Numpy, es posible comprobar la solución con la obtenida por HOFEM para comprobar la eficacia del código anterior, en este caso, las matrices se generaron correctamente.

4.2 Paso2 – Resolución del problema de Elementos Finitos mediante Petsc4Py

Llegados a este punto, era necesario conectar ambos códigos, el de Petsc4Py para poder resolver el sistema de ecuaciones a través de diferentes *solvers*, y el código de Python que contenía las matrices de HOFEM en formato de array de Numpy. Sin embargo, esto entraña una dificultad añadida, ya que, como se ha comentado anteriormente, las matrices y vectores de PETSc son distintas a las generadas por Numpy.

En una primera aproximación, se intentó crear las matrices de PETSc como en el primer código, pero, en vez de con números aleatorios, con los elementos de las matrices de HOFEM. Sin embargo, este primer intento resultó en fracaso, ya que no es posible referenciar juntas una matriz de tipo PETSc y un array de NumPy, ya que Petsc4Py no es capaz de identificar los arrays de Numpy.

De este modo, se optó por generar las matrices de PETSc del mismo tamaño que las obtenidas de HOFEM, pero completamente vacías, y posteriormente, mediante el uso de un bucle, recorrer cada una de las posiciones de esa matriz vacía, y asignarle el valor correspondiente de la matriz de HOFEM. A continuación se detalla el código resultante:

```
## creamos matriz sparse
A = PETSc.Mat()
A.create(PETSc.COMM_WORLD)
A.setSizes([max_IRN, max_IRN])
A.setType('aij') # matriz sparse

##funcion obligada para poder llamar al resolvedor mas tarde (PETSc FAQ 3.3)
aux = A.setUp()

for k in range(0,max_IRN):
        A.setValue(k,k,A_man[k,k])

# communicate off-processor values
# and setup internal data structures
# for performing parallel operations
A.assemblyBegin()
A.assemblyEnd()
```

Figura 25. Código Python/Petsc4Py para las matrices de HOFEM

Una vez terminado un código similar para el vector RHS (right-hand side), se procedió a la comparación entre distintos métodos.

```
# vectores rhs pre-solucion
x, b = A.getVecs()
# Inicializacion
x.set(0)

for k in range(0,max_IRN):
        b.setValue(k,RHS_man[k])

b.assemblyBegin()
b.assemblyEnd()
```

Figura 26. Código Python/Petsc4Py para las matrices de HOFEM (II)

Siendo 'b' el vector RHS y 'x' la solución del sistema de ecuaciones. En Petsc4Py es necesario inicializar vectores y matrices, de modo que en este caso, el vector que proporciona la solución del sistema se inicializó a cero en todas sus posiciones, para posteriormente rellenarlo con los valores obtenidos tras la utilización del solver correspondiente.

Es importante destacar el hecho de que PETSc posee funciones de lectura y escritura en ficheros al igual que Python, si bien no se trata de las mismas funciones, y su funcionamiento no es tan intuitivo como pueda serlo el de las contenidas en NumPy. La función más utilizada en este aspecto es *MatLoad()*, si bien es altamente eficiente y permite leer matrices de gran tamaño, tiene como requisito obligatorio que dichas matrices estén escritas en formato binario.

4.3 Comparación entre distintos solvers para un problema de Elementos Finitos.

Se utilizaron seis métodos distintos para llevar a cabo este estudio, además de la solución proporcionada por HOFEM a través de MUMPS. A continuación se muestra un diagrama con los diferentes métodos utilizados y su tipo.

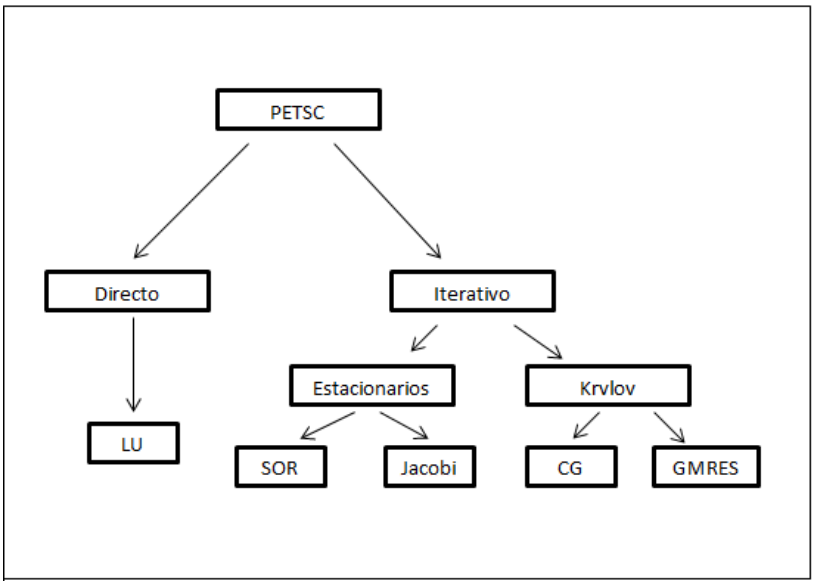


Figura 27. Diagrama de bloques de los distintos solvers de Python utilizados

Los resultados que se muestran a continuación se han obtenido después de una ponderación de diez muestras. Es importante destacar el hecho de que no existen diferencias sustanciales entre el *accuracy*(diferencia entre la solución analítica obtenida con Matlab y la solución que devuelve Python)de la solución debido a la dimensión de las matrices del problema tratado ya que la solución se aproxima a la obtenida con HOFEM con un cero numérico.

A continuación se muestran los resultados obtenidos en formato de gráfica y en formato tabla, dado el principal objetivo del presente Trabajo Fin de Grado, dichos resultados hacen referencia al tiempo de resolución de cada uno de los *solvers* mencionados para las distintas matrices proporcionadas por HOFEM.

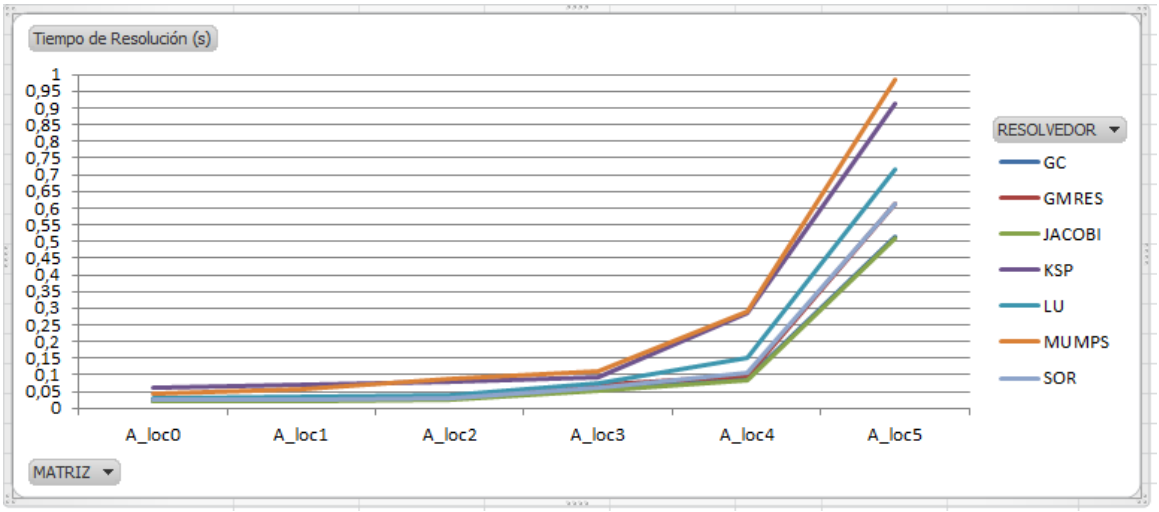


Figura 28. Tiempo de resolución para distintos solvers y matrices.

	CG	GMRES	JACOBI	KSP	LU	SOR	MUMPS
A_loc0	0,0229517	0,0265638	0,0228429	0,0594471	0,0292014	0,0235140	0,045
A_loc1	0,0265717	0,0315499	0,0220513	0,0704789	0,0332200	0,0249195	0,059
A_loc2	0,0296092	0,0341208	0,0245690	0,0808389	0,0387811	0,0280690	0,088
A_loc3	0,0548622	0,0684504	0,0532903	0,0908110	0,0730598	0,0597706	0,113
A_loc4	0,0880198	0,0973396	0,0839691	0,2837916	0,1510701	0,1074088	0,29
A_loc5	0,5135802	0,6135741	0,5104510	0,9137514	0,7169429	0,6141739	0,984

Tabla 5.Tiempo de resolución para distintos solvers y matrices.

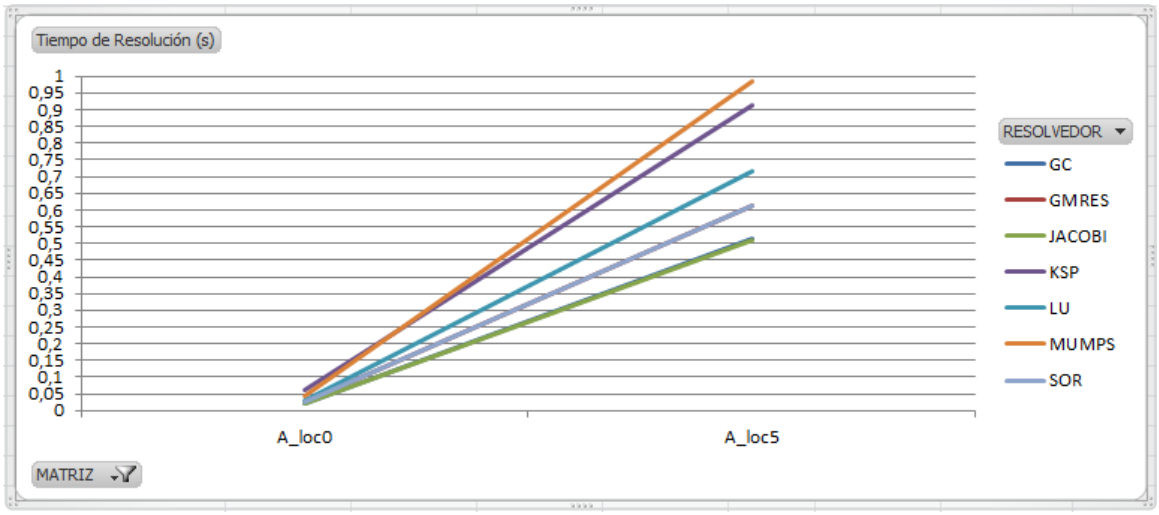


Figura 29. Tiempo de Resolución para matrices muy dispares en tamaño.

Como podemos observar en los datos obtenidos, en general, los *solvers* iterativos son más rápidos que el *solver* directo (LU), aunque el KSP no se ajuste a esta afirmación, es posible que se deba a las características de la matriz de Elementos Finitos.

A la vista de los resultados obtenidos, todos los *solvers* analizados tienen mejores prestaciones que MUMPS, pero sería necesario un estudio más detallado para afirmar categóricamente que merece la pena utilizar los resolvedores de Python antes que los de MUMPS, además de resolver diferentes problemas electromagnéticos para estudiar posibles inconvenientes de utilización de los *solvers* iterativos por temas de rendimiento o *accuracy* al realizar cambios en la estructura electromagnética a simular. Es importante señalar el hecho de que MUMPS está optimizado para tamaños de matrices muy grandes, del orden de centenares de miles de incógnitas, que no se han utilizado durante el presente Trabajo Fin de Grado.

Capítulo 5

Conclusiones y Líneas Futuras

5.1 Conclusiones

El lenguaje de programación Python se está convirtiendo durante los últimos años en un referente para la computación numérica gracias al desarrollo de numerosas herramientas para el cálculo científico como las que se han descrito en este documento.

El objetivo de este proyecto ha sido estudiar el uso de diferentes *solvers* de Python para su utilización en sistemas de ecuaciones correspondientes a problemas de Elementos Finitos y su interconexión con la *suite software* de HOFEM. Este *software* se encuentra desarrollado en FORTRAN desde el año 2010, en el que es característica su alta eficiencia computacional, pero es poco flexible, de forma que cambiar fragmentos de este código es costoso. Python destaca por su flexibilidad y por tener una comunidad creciente de programadores, de forma que sus implementaciones son cada vez más eficientes y rápidas.

CAPÍTULO 5: CONCLUSIONES Y LÍNEAS FUTURAS

En una primera instancia, se optó por el uso de una herramienta de encapsulación de código FORTRAN para su utilización a través de la interfaz de Python, f2Py. A través de una serie de pruebas, se fueron desarrollando distintas versiones de código capaces de replicar las principales características del código inicial, y que se han descrito a lo largo de este documento, hasta que diversos problemas obligaron a estudiar otras posibles vías de acción.

Este nuevo enfoque incluía la implementación de una interfaz basada en el paso de mensajes por medio de escritura en disco. Se incorporó el uso de una nueva herramienta de Python, petsc4Py, que es la implementación en este lenguaje de PETSc (Portable Extensible Toolkit for Scientific Computation). Se realizaron una nueva serie de pruebas, tanto para el manejo de matrices de gran tamaño en Python como para su posterior incorporación a petsc4Py y la aplicación de diferentes *solvers* para la resolución del sistema de ecuaciones correspondiente a un problema de Elementos Finitos proporcionado por HOFEM.

Por último se realizó un estudio comparativo entre diferentes *solvers* de los incluidos en la herramienta petsc4Py y MUMPS en FORTRAN, atendiendo principalmente al tiempo de resolución de cada uno de ellos para diferentes matrices. Ese estudio reveló una posible utilización de Python en la resolución de este tipo de problemas, si bien es cierto que serán necesarias consecuentes pruebas de concepto para afirmarlo categóricamente.

5.2 Futuras Líneas de Implementación

El desarrollo de futuras líneas de trabajo en este ámbito, pasa por resolver los problemas descritos en el capítulo 3 de este documento con la consecuente posibilidad de utilizar el código FORTRAN de HOFEM usando *f2py* como una pasarela entre ambos lenguajes, con la ventaja añadida de usar la interfaz de Python, en teoría, mucho más flexible y optimizable que la de FORTRAN.

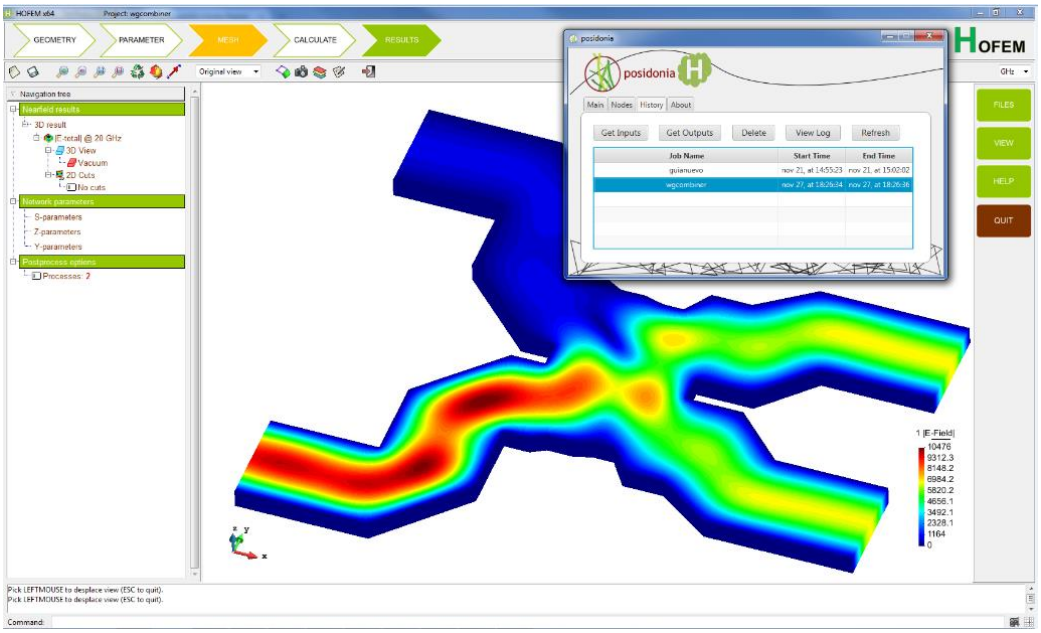


Figura 30. Simulación HOFEM de un problema de electromagnetismo.

Si bien la resolución del problema descrito anteriormente es una de las tareas a desarrollar, es muy importante la realización de un estudio comparativo más detallado para estudiar la viabilidad de sustituir MUMPS por los *solvers* proporcionados por Python, así como su utilización en diferentes problemas electromagnéticos para estudiar posibles inconvenientes en su utilización por temas de rendimiento al realizar cambios significativos en la estructura electromagnética a simular.

5.3Herramientas Objeto de Estudio

A continuación se expone la utilización de un módulo extra para crear contenedores intermedios de código *.f90* que permitan un programa con características de FORTRAN 'modernas' como los tipos derivados, y su posterior compilación mediante *f2py.*

5.2.1 f2py_wrapper_gen – Encapsulado de Código Fuente de FORTRAN 90

Este módulo se utiliza para escribir archivos de encapsulado intermedio de FORTRAN 90 que permitan un código que haga uso de tipos derivados para ser utilizados mediante *f2py.*

Todas las rutinas que aceptan argumentos de tipos derivados son encapsuladas por rutinas equivalentes que, en cambio, reciban arrays de enteros como argumentos opacos. La función intrínseca *transfer()* de FORTRAN se utiliza para convertir estos argumentos en punteros a los tipos derivados [26]. El uso de arrays de enteros de 12 posiciones hace que de este enfoque portable, a través de la mayoría de compiladores FORTRAN disponibles actualmente. De esta manera podemos acceder a las estructuras subyacentes de FORTRAN desde Python.

Las rutinas *initialise()* y *finalise()* son tratadas de manera especial; para la inicialización, un puntero de tipo derivado se asigna antes de invocar a la rutina de encapsulado, y se devuelve una referencia opaca a este nuevo tipo derivado. En la finalización, el puntero del tipo derivado subyacente se desasigna después de las declaraciones de las rutinas de encapsulado.

Se generan rutinas adicionales para acceder a los valores de los atributos que utilicen tipos derivados. Para argumentos de tipo escalar, se crean rutinas de tipo *get* y *set*, mientras que para arrays se utiliza una única rutina que devuelve el tamaño, la ubicación de memoria y el tipo de array de salida. Estas rutinas son utilizadas por *quippy.oo_fortran*[26] en la construcción de la capa de encapsulado orientada a objetos.

5.2.2Sfepy: Simple Finite Elements in Python

Por último, se menciona un software (aún en desarrollo) para el tratamiento de problemas de Elementos Finitos con código en Python.

Sfepy es un software para la resolución de sistemas de ecuaciones diferenciales acopladas (PDE) por el método de Elementos Finitos (FEM) en una, dos y tres dimensiones. Puede ser visto tanto como un solver PDE cómo un paquete Python que puede utilizarse para crear aplicaciones personalizadas. La palabra '*simple*' hace referencia a que los problemas de Elementos Finitos complejos pueden ser codificados con mucha facilidad y rapidez.

Figura 31. Logotipo de Sfepy

Este software puede utilizar una gran variedad de términos para construir las PDE que hay que resolver [11]. En la página web del proyecto es posible encontrar una serie de ejemplos para ayudar en el comienzo del manejo de esta herramienta, así como ejemplos avanzados y un tutorial de funcionamiento. Actualmente se está trabajando para lograr su funcionamiento en paralelo, así como en un soporte preliminar para el análisis isogeométrico.

El código está escrito casi completamente en Python, con excepción de las rutinas que demandan más tiempo, las cuales están escritas en C y encapsuladas mediante *Cython* o escritas directamente en *Cython.*

Sfepy es un software libre publicado bajo la licencia 'New BSD'. Está basado en *Numpy* y *Scipy* (una excelente colección de herramientas para cálculo numérico en Python). Se trata de un software multiplataforma capaz de operar en la mayoría de sistemas operativos disponibles en el mercado, como Linux, Mac OS o Windows.

Fue desarrollado originalmente como un marco flexible para implementar y poner a prueba rápidamente los métodos matemáticos desarrollados durante otros proyectos de investigación. Ha evolucionado, sin embargo, a una (todavía pequeña) herramienta multifuncional para resolución de problemas de Elementos Finitos. Muchas de esas funcionalidades se han implementado para que puedan ser usadas para construir las PDEs.

Presupuesto

En este capítulo se justificarán los costes de realización de este Trabajo Fin de Grado una vez que se ha finalizado, los cuales se dividirán en:

1. Coste de las horas empleadas.

La duración del presente proyecto ha sido de alrededor de 7 meses, lo que significa aproximadamente un tiempo de 600 horas invertidas, repartidas de la siguiente manera:

ACTIVIDAD	HORAS
Aprendizaje del uso de la herramienta F2PY	30
Obtención de documentación	50
Interconexión HOFEM-Python	140
Aprendizaje del uso de la herramienta Petsc4Py	30
Obtención de documentación	40
Estudio comparativo entre solvers	120
Realización de la memoria	190
Total	**600**

ACTIVIDAD	PRECIO/HORA	HORAS
Coste de las horas empleadas por el alumno	5.5€	600
Coste de las horas empleadas por el tutor del proyecto	30€	70
Total		**5400€**

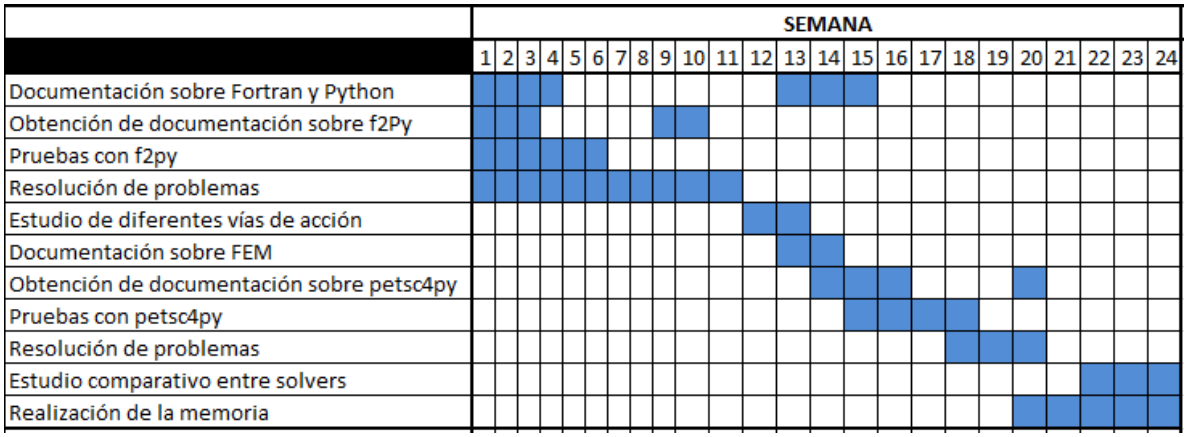
2. Coste del software empleado.

Los lenguajes de programación empleados durante el presente proyecto o bien se encuentran gratuitamente en la red por ser de carácter *open source* o bien han sido proporcionados por la universidad, además de la *suite software* HOFEM, la cual no se ha construido expresamente para la realización del proyecto por encontrarse ya en funcionamiento y uso.

Por tanto, el presupuesto total del presente proyecto es íntegramente el coste de las horas empleadas en su realización, cuya suma asciende a la cantidad de **5400€.**

Planificación del Proyecto

En esta página se muestra el diagrama de Gantt del proyecto, resultante de un trabajo previo de planificación.



Glosario

CST	*Computer Simulation Technology*
FORTRAN	FORmulaTRANslatingSystem
API	*ApplicationProgramming Interface*
ASCII	*American Standard CodeforInformationInterchange*
ANSI	*American NationalStandardsInstitute*
CWI	*CentrumWiskunde&Informatica*
PETSc	*Portable Extensible ToolkitforScienticComputation*
MPI	*MessagePassing Interface*
MPI4PY	*MessagePassing Interface forPython*
MUMPS	*MultifrontalMassivelyParallelSparseDirectSolver*
F2PY	*FORTRAN toPython Interface Generator*
HOFEM	*Higher Order Finite Element Method*
OOP	*Object Oriented Programming*
FEM	*Finite Elements Method*

Referencias

[1] “Sitio oficial de desarrollo de CST – ComputerSimulationTechnology”, https://www.cst.com/, consultado en sep. 2015

[2] “Sitio oficial de desarrollo de FEKO”, https://www.feko.info/, consultado en sep. 2015

[3] “Sitio oficial de desarrollo de HFSS”, http://www.ansys.com/Products/Simulation+Technology/Electronics/Signal+Integrity/ANSYS+HFSS, consultado en sep. 2015

[4] Jianming, Jin. 2014. The Finite Element Method in Electromagnetics (3rd ed.). Wiley-IEEE Press.

[5] D. García-Doñoro, I. Martínez-Fernández, L. E. García-Castillo, y M. Salazar-Palma, “HOFEM: A higher order finite element method electromagnetic simulator,” in 12th International Workshop on Finite Elements for Microwave Engineering, Mount Qingcheng, Chengdu, China, May 2014.

[6] D. García-Doñoro, “A new software suite for electromagnetics,” Jul. 2014, tesis doctoral. Universidad Carlos III de Madrid. Calificación: Sobresaliente cum laude.

[7] “Wiki de FORTRAN”, http://fortranwiki.org/fortran/show/HomePage, consultado en junio 2015

[8] “Ranking de supercomputadoras más rápidas del mundo”, http://www.top500.org/lists/2015/06/, consultado en junio 2015.

[9] “*Preliminary Report, Specifications for the IBM Mathematical FORmula TRANSlating System, FORTRAN*”, http://archive.computerhistory.org/resources/text/Fortran/102679231.05.01.acc.pdf

[10] Reid, John. 2003. The New Features of FORTRAN 2003. Final Committee Draft (FCD) of the FORTRAN 2003 standard.

[11] Reid, John 2008. The New Features of FORTRAN 2008. ACM FORTRAN Forum 27(2), 8-21.

[12] Venners, Bill 2003. The Making of Python. A Conversation with Guido van Rossum, Part I.

[13] “Sitio oficial de Python”, https://docs.python.org/2/license.html,consultado en junio 2015.

[14] “Sitio oficial de desarrollo de SciPy”, http://scipy.org/, consultado en junio 2015

[15] “Sitio oficial de desarrollo de PETSC”, http://www.mcs.anl.gov/petsc/index.html, consultado en julio 2015

[16] D. M. Beazley 2000. Scientific Computing with Python. Department of Computer Science, University of Chicago.

[17] “Sitio oficial de desarrollo de Mpi4Py”, https://mpi4py.scipy.org/docs/usrman/index.html, consultado junio 2015.

[18] “Sitio oficial de desarrollo de PyMumps”, https://pypi.python.org/pypi/PyMUMPS, consultado junio 2015.

[19] PythonThreads Interview to Pearu Peterson 2006. Once I learned about Python, I stopped triyng out differente languages

[20] “Sitio oficial de desarrollo de TCL-TK”, https://www.tcl.tk/, consultado en sep. 2015.

[21] “Sitio oficial de desarrollo de GiD”, http://www.gidhome.com/, consultado en sep. 2015.

[22] “Estándares MPI”, http://www.mcs.anl.gov/research/projects/mpi/, consultado en sep. 2015.

[23]Pearu Peterson, Joaquin R. R. A Martins y Juan J. Alonso. Fortran to Python Interface Generator with an Application to Aerospace Engineering, January 2001.

[24] “Sitio oficial de FORTRAN 90 – Interfaz con Python”, http://www.fortran90.org/src/best-practices.html#interfacing-with-python, consultado en junio 2015.

[25] Timothy A. Davis. Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms), University of Florida, Gainesville, Florida, 2006.

[26] “Documentación de la función MatCreate()”,http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/Mat/MatCreate.html#MatCreate, consultado en julio 2015.

[27] “Sitio oficial de Quippy-FORTRAN”, http://libatoms.github.io/QUIP/intro.html#pletzer2008, consultado en agosto 2015.

[28] “Sitio oficial del proyecto SfePy”, http://sfepy.org/doc-devel/terms_overview.html#term-overview, consultado en agosto 2015.

Bibliografía

J.W. Backus, R.J. Beeber, S. Best, R. Goldberg, L.M. Haibt, H.L. Herrick, R.A. Nelson, D. Sayre, P.B. Sheridan, H.J. Stern, I. Ziller, R.A. Hughes, and R. Nutt, The FORTRAN automatic coding system. Pages 188-198. In *Proceedings Western Joint Computer Conference*, Los Angeles, California, February 1957.

Knowlton, Jim (2009). *Python*.tr: Fernández Vélez, María Jesús (1 edición). Anaya Multimedia-Anaya Interactiva.

Hans Petter Langtangen (2014). A Primer on Scientific Programming with Python (Fourth Edition).

ČERTÍK, Ondřej et al. *Fortran 90*

DOWNLING, John. *Interfacing Python with FORTRAN*

R. Cimrman. SfePy–write your own FE application. In P. de Buyl and N. Varoquaux, editors, Proceedings of the 6th European Conference on Python in Science (EuroSciPy 2013)

Jianming, Jin. 2014. The Finite Element Method in Electromagnetics (3rd ed.). Wiley-IEEE Press.

D. García-Doñoro, I. Martínez-Fernández, L. E. García-Castillo, y M. Salazar-Palma, "HOFEM: A higher order finite element method electromagnetic simulator," in 12th International Workshop on Finite Elements for Microwave Engineering, Mount Qingcheng, Chengdu, China, May 2014.

D. García-Doñoro, "A new software suite for electromagnetics," Jul. 2014, tesis doctoral. Universidad Carlos III de Madrid. Calificación: Sobresaliente cum laude.

Timothy A. Davis. Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms), University of Florida, Gainesville, Florida, 2006.